FI MU

Fakulta informatiky
Masarykova univerzita

Automaty a formální jazyky I

Ivana Černá
Mojmír Křetínský
Antonín Kučera

Učební text FI MU **verze 1.3.1**

 únor 2014

Obsah

Kapitola 1

Jazyk a jeho konečná reprezentace

V této kapitole budou zavedeny základní pojmy, které jsou předmětem zkoumání teorie formálních jazyků.

1.1 Abeceda a jazyk

Abecedou se rozumí libovolná konečná množina Σ, jejíž prvky nazýváme *znaky* (případně také *písmena* nebo *symboly*) abecedy. Příkladem abecedy je třeba množina $\{a, b\}$, množina číslic $\{0, 1, \ldots, 9\}$, nebo prázdná množina $\emptyset$.

Slovo (též *řetězec*) v nad abecedou Σ je libovolná konečná posloupnost znaků této abecedy (např. $aabb$ je slovo nad abecedou $\{a, b\}$). Počet členů této posloupnosti v značíme $|v|$ a nazýváme délkou slova, počet výskytů znaku a ve slově v značíme $\#_a(v)$ (např. $\#_b(abaaba) = 2$). Prázdné posloupnosti znaků odpovídá tzv. *prázdné slovo*, označované ε, které má nulovou délku.

Množinu všech slov nad abecedou Σ značíme Σ^*, množinu všech neprázdných slov Σ^+. Platí např.

$$
\begin{aligned}
\{a\}^* &= \{\varepsilon, a, aa, aaa, aaaa, \ldots\} \\
\{a\}^+ &= \{a\}^* \setminus \{\varepsilon\} \\
\{0,1\}^* &= \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, \ldots\}
\end{aligned}
$$

Definitoricky dále klademe $\emptyset^* = \{\varepsilon\}$ a $\emptyset^+ = \emptyset$. Na každá dvě slova u, v lze aplikovat binární operaci *zřetězení*, označovanou '.', která je definována předpisem $u.v = uv$. Např. zřetězením slov $abba$ a bba obdržíme slovo $abbabba$. Operace zřetězení je asociativní, tj. $u.(v.w) = (u.v).w$ pro libovolná slova u, v, w. Dále se ε chová jako jednotkový prvek, tj. $u.\varepsilon = \varepsilon.u = u$ pro libovolné slovo u. V dalším textu budeme znak '.' v zápisu zřetězení obvykle vynechávat. Každé neprázdné slovo nad abecedou lze získat zřetězením symbolů této abecedy – odtud *řetězec* jako ekvivalent pojmu slovo.

Pro snadnější specifikaci jazyků je výhodné zavést unární operaci *i-té mocniny* slova, která je definovaná induktivně pro každé $i \in \mathbb{N}_0$ takto: nechť Σ je libovolná abeceda, $u \in \Sigma^*$ libovolné slovo. Pak

- $u^0 = \varepsilon$,
- $u^{i+1} = u.u^i$.

Např. $(abc)^3 = abcabcabc$ (kulaté závorky slouží jako metasymboly; nejsou součástí slova, pouze pomáhají vymezit argument operace).

Slovo u je *podslovem* slova v, jestliže existují slova x, y taková, že $v = xuy$. Pokud navíc $x = \varepsilon$, říkáme že slovo u je *předponou* (nebo také *prefixem*) slova v, což značíme $u \preceq v$ ($\preceq$ je tedy částečným uspořádáním na Σ^*); je-li v tomto případě ještě navíc $u \neq v$ (tj. $y \neq \varepsilon$), pak klademe $u \prec v$. Je-li $y = \varepsilon$ (nyní již bez omezujících požadavků na x), nazveme u *příponou* (*sufixem*) slova v. Například aa je předponou $aabab$, zatímco 11 není ani podslovem slova 01010. Konečně pro každé $k \geq 1$ celé a slovo v značíme zápisem $k : v$ předponu slova v délky (nejvýše) k a dejinujeme $k : v = u$, pokud existuje w takové, že $v = uw$ a $|u| = k$; v opačném případě klademe $k : v = v$. Zejména tedy pro neprázdné slovo v zápis $1 : u$ znamená první znak slova v.

Jazyk nad abecedou Σ je libovolná množina slov nad Σ (jazyky nad Σ jsou tedy právě podmnožiny Σ^*). Např. $\{10, 1, 011101\}$ je jazyk nad abecedou $\{0, 1\}$, prázdná množina je jazyk nad libovolnou abecedou, atd. Jazyky mohou ovšem být i nekonečné, např. $\{w \in \{a, b\}^* \mid \#_a(w) = \#_b(w)\}$ je jazyk nad abecedou $\{a, b\}$ obsahující všechna slova, ve kterých se a i b vyskytují ve stejném počtu (tedy např. $aababb, \varepsilon$ jsou prvky tohoto jazyka, zatímco $a, abbaa$ nikoliv).

1.1.1 Operace nad jazyky

V této části zavedeme některé operace nad jazyky, které se v dalších kapitolách ukáží jako velmi důležité. Je třeba důsledně rozlišovat mezi operacemi nad *slovy* a operacemi nad *jazyky*, i když se některé z nich (jako třeba zřetězení nebo mocnina) značí v obou případech stejně. Na základě typu parametrů bude vždy jasné, zda se jedná o operaci nad slovy nebo jazyky.

Nechť L je libovolný jazyk nad abecedou Σ a K libovolný jazyk nad abecedou Δ. Jelikož L i K jsou množiny, můžeme aplikovat standardní množinové operace *sjednocení*, *průnik* a *rozdíl*. Výsledkem je vždy jazyk nad abecedou $\Sigma \cup \Delta$. Dále definujeme:

- *Zřetězením* jazyků K a L je jazyk $K.L = \{uv \mid u \in K, v \in L\}$ nad abecedou $\Sigma \cup \Delta$. Podle definice zejména platí $\emptyset.M = M.\emptyset = \emptyset$ a $\{\varepsilon\}.M = M.\{\varepsilon\} = M$ pro libovolný jazyk M. Operace zřetězení jazyků je také zřejmě asociativní.
- i*-tá mocnina* jazyka L je definována induktivně pro každé $i \in \mathbb{N}_0$:
  1. $L^0 = \{\varepsilon\}$
  2. $L^{i+1} = L.L^i$

  Zejména tedy $\emptyset^0 = \{\varepsilon\}$, $\emptyset^i = \emptyset$ pro libovolné $i \in \mathbb{N}$ a $\{\varepsilon\}^j = \{\varepsilon\}$ pro libovolné $j \in \mathbb{N}_0$.
- *Iterace* jazyka L je jazyk $L^* = \bigcup_{i=0}^{\infty} L^i$.
- *Pozitivní iterace* jazyka L je jazyk $L^+ = \bigcup_{i=1}^{\infty} L^i$. Obecně není pravda, že $L^+ = L^* \setminus \{\varepsilon\}$; tato rovnost platí právě tehdy, když L neobsahuje ε.
- *Doplněk* jazyka L je jazyk co–$L = \Sigma^* \setminus L$.

- *Zrcadlovým obrazem* (též reverzí) slova $w = a_1 \dots a_n$ nazýváme slovo $w^R = a_n \dots a_1$ ($\varepsilon^R = \varepsilon$). *Zrcadlový obraz jazyka* L definujeme jako $L^R = \{w^R | w \in L\}$.
- *Substituce* f je zobrazení abecedy Σ do množiny podmnožin množiny Δ^*, tj. f přiřazuje každému $a \in \Sigma$ jazyk $f(a) \subseteq \Delta^*$. Zobrazení f je rozšířeno na slova takto:
  1. $f(\varepsilon) = \varepsilon$,
  2. $f(xa) = f(x)f(a)$

  a na jazyky f rozšíříme takto: pro každé $L \subseteq \Sigma^*$ klademe $f(L) = \bigcup_{x \in L} f(x)$. Substituci f nazveme *nevypouštějící*, jestliže pro žádné $a \in \Sigma$ jazyk $f(a)$ neobsahuje ε.
- Speciálním případem substituce je *homomorfismus* h, který definujeme jako substituci, u níž $h(a)$ obsahuje jediné slovo pro každé $a \in \Sigma$. V tomto případě obvykle $h(a)$ považujeme za slovo a nikoli jednoprvkovou množinu obsahující toto slovo. Je-li navíc $h(a) \neq \varepsilon$ pro všechna $a \in \Sigma$, říkáme, že h je *nevypouštějící* (též ε-free).
- Je-li h homomorfismus, pak definujeme *inverzní homomorfní obraz* jazyka L jako $h^{-1}(L) = \{x | h(x) \in L\}$ a pro slova definujeme *inverzní homomorfismus* jako $h^{-1}(w) = \{x | h(x) = w\}$.

V souvislosti s operací iterace je dobré si povšimnout, že symbolem Σ^* jsme v předchozí části označili množinu všech slov nad abecedou Σ. Jelikož samotné Σ je možné chápat jako jazyk nad Σ (obsahující právě všechna slova délky jedna), lze zápis Σ^* interpretovat také jako iteraci jazyka Σ. Tato nejednoznačnost však není na závadu; snadno se vidí, že iterací Σ obdržíme právě jazyk obsahující všechna slova nad Σ. Ze stejného důvodu je nezávadná dvojznačnost zápisu Σ^+.

Nechť $\mathcal{L}$ je třída jazyků a o je n-ární operace na jazycích. Řekneme, že $\mathcal{L}$ je *uzavřená* na o, pokud pro libovolné jazyky $L_1, \dots, L_n$ patřící do $\mathcal{L}$ platí, že také jazyk $o(L_1, \dots, L_n)$ patří do $\mathcal{L}$.

Příklad 1.1. *Nechť $\mathcal{L}$ je množina tvořená jazyky $L_i = \{a^i\}$, kde $i \in \mathbb{N}_0$. Pak $\mathcal{L}$ je uzavřena na zřetězení a mocninu, ale není uzavřena na doplněk, sjednocení, průnik, rozdíl a iteraci . Příklady substitucí a homomorfismů budou uvedeny v relevantních partiích tohoto textu.*

1.2 Konečná reprezentace jazyka

Většina jazyků, kterými se budeme zabývat v tomto textu, bude obsahovat nekonečný počet slov. V této souvislosti velmi přirozeně vyvstávají některé důležité otázky.

Jedna z nich se nabízí okamžitě – jak konečným způsobem reprezentovat (obecně nekonečný) jazyk, tj. specifikovat množinu všech jeho slov (tzv. problém *konečné reprezentace*). Pokud by jazyk byl konečný (tj. obsahoval pouze konečně mnoho slov), pak vystačíme s výčtem všech jeho slov. V případě jazyka nekonečného je problém jeho konečné reprezentace velice podstatný. Konečná reprezentace by měla být (formálně vzato) opět řetězcem symbolů (nad jistou abecedou), který budeme vhodně interpretovat tak, aby danému jazyku odpovídala nějaká (ne nutně jediná) konkrétní konečná reprezentace; obráceně pak, aby každé konkrétní konečné reprezentaci odpovídal *jediný* konkrétní jazyk. Takovými reprezentacemi pro nás v dalším textu budou zejména tzv. gramatiky, které zavedeme v této kapitole, a automaty, které budeme definovat později.

V návaznosti na právě řečené vystává další otázka, a to zda pro každý jazyk existuje jeho konečná reprezentace. Zde lze po krátkém zamyšlení očekávat negativní odpověď: množina všech řetězců nad libovolnou abecedou je spočetně nekonečná, ale systém všech podmnožin spočetně nekonečné množiny je množina nespočetná. Jelikož bychom měli být schopni zapsat jakoukoli definici konečné reprezentace jako nějaký řetězec symbolů, lze (alespoň na intuitivní úrovni) očekávat, že dostaneme jen spočetně mnoho konečných reprezentací, a tedy že existuje mnohem víc jazyků, než konečných reprezentací (formálněji jsou tyto úvahy formulovány a dokázány až po definici pojmu gramatiky v tvrzení 1.4).

Konečně si lze položit otázku, jaká je struktura, vlastnosti, ... těch tříd jazyků, pro něž existují konečné reprezentace (a to v souvislosti s "typem"této reprezentace), jak lze tyto třídy charakterizovat atp. Těmto problémům je věnována převážná část tohoto učebního textu.

1.2.1 Pojem gramatiky

Definice 1.2. *Gramatika* $\mathcal{G}$ je čtveřice (N, Σ, P, S), kde

- N je neprázdná konečná množina *neterminálních symbolů* (stručněji: *neterminálů*).
- Σ je konečná množina *terminálních symbolů (terminálů)* taková, že $N \cap \Sigma = \emptyset$. Sjednocením N a Σ obdržíme množinu *všech symbolů* gramatiky, kterou obvykle označujeme symbolem V.
- $P \subseteq V^*NV^* \times V^*$ je konečná množina *pravidel*. Pravidlo (α, β) obvykle zapisujeme ve tvaru $\alpha \to \beta$ (a čteme jako "α přepiš na β").
- $S \in N$ je speciální *počáteční* neterminál (nazývaný také *kořen gramatiky*).

Podmínka kladená na tvar pravidel $\alpha \to \beta$ pouze požaduje, aby α (tzv. *levá strana* pravidla) obsahovala alespoň jeden neterminál; na β (*pravou stranu* pravidla) nejsou obecně kladeny žádné požadavky – specielně, může být i prázdným slovem ε.

Každá gramatika $\mathcal{G} = (N, \Sigma, P, S)$ určuje binární relaci $\Rightarrow_{\mathcal{G}}$ *přímého odvození* na množině V^* definovanou takto: $\gamma \Rightarrow_{\mathcal{G}} \delta$ právě když existuje pravidlo $\alpha \to \beta \in P$ a slova $\eta, \varrho \in V^*$ taková, že $\gamma = \eta\alpha\varrho$ a $\delta = \eta\beta\varrho$. V dalších kapitolách budeme rovněž potřebovat tyto relace na V^*:

- relaci *odvození v k krocích* (k-násobné složení relace $\Rightarrow_{\mathcal{G}}$) značíme $\overset{k}{\Rightarrow}_{\mathcal{G}}$ a definujeme pro každé $k \in \mathbb{N}_0$ induktivně takto:
  - $\overset{0}{\Rightarrow}_{\mathcal{G}}$ je identická relace
  - $\overset{k+1}{\Rightarrow}_{\mathcal{G}} = \overset{k}{\Rightarrow}_{\mathcal{G}} \circ \Rightarrow_{\mathcal{G}}$
- relaci *odvození v nejvýše k krocích*, kterou značíme $\overset{\leq k}{\Rightarrow}_{\mathcal{G}}$ a definujeme pro každé $k \in \mathbb{N}_0$ předpisem

$$\overset{\leq k}{\Rightarrow}_{\mathcal{G}} = \bigcup_{i=0}^{k} \overset{i}{\Rightarrow}_{\mathcal{G}}$$

- relaci *odvození*, ktrerou značíme $\Rightarrow_{\mathcal{G}}^{*}$ a definujeme předpisem

$$\Rightarrow_{\mathcal{G}}^{*} = \bigcup_{i=0}^{\infty} \overset{i}{\Rightarrow}_{\mathcal{G}}$$

  Tedy $\Rightarrow_{\mathcal{G}}^{*}$ je reflexivní a tranzitivní uzávěr $\Rightarrow_{\mathcal{G}}$.
- Relace *netriviálního odvození*, kterou značíme $\Rightarrow_{\mathcal{G}}^{+}$, je definována takto:

$$\Rightarrow_{\mathcal{G}}^{+} = \bigcup_{i=1}^{\infty} \overset{i}{\Rightarrow}_{\mathcal{G}}$$

  Relace $\Rightarrow_{\mathcal{G}}^{+}$ je tedy tranzitivní uzávěr relace $\Rightarrow_{\mathcal{G}}$.

V dalším textu budeme index $\mathcal{G}$ u výše uvedených relací obvykle vynechávat, bude-li z kontextu patrné, o kterou gramatiku se jedná.

Prvky množiny V^*, které lze odvodit z počátečního neterminálu, nazýváme *větnými formami* gramatiky $\mathcal{G}$. Přesněji, $\alpha \in V^*$ je větná forma právě když $S \Rightarrow^* \alpha$. Větná forma, která neobsahuje žádné neterminály, se nazývá *věta*. Množina všech vět tvoří jazyk *generovaný* gramatikou $\mathcal{G}$, označovaný jako $L(\mathcal{G})$:

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

Gramatiky $\mathcal{G}_1$ a $\mathcal{G}_2$ nazveme *jazykově ekvivalentní* (dále jen ekvivaletní), právě když generují tentýž jazyk, tj. $L(\mathcal{G}_1) = L(\mathcal{G}_2)$.

Příklad 1.3. *Nechť* $\mathcal{G} = (\{S, A, B\}, \{a, b\}, P, S)$, *kde*

$$\begin{aligned} P = \{ \; & S \to ABS, \\ & S \to \varepsilon, \\ & AB \to BA, \\ & BA \to AB, \\ & A \to a, \\ & B \to b \; \} \end{aligned}$$

Pak např. $AaBAbBS$ *je větná forma* $\mathcal{G}$, *zatímco* ABb *nikoliv. Jazyk* $L(\mathcal{G})$ *vypadá takto:* $L(\mathcal{G}) = \{u \in \{a, b\}^* \mid \#_a(u) = \#_b(u)\}$.

V následujícím textu budeme dodržovat tyto konvence týkající se značení:

- Počáteční neterminál (kořen) gramatiky obvykle značíme symbolem S.
- Neterminální symboly jsou označovány velkými písmeny (obvykle ze začátku) latinské abecedy $(A, B, C, \dots)$
- Terminální symboly obvykle značíme malými písmeny ze začátku latinské abecedy $(a, b, c, \dots)$.
- Řetězce, které jsou složeny výhradně z terminálních symbolů (tzv. terminální řetězce) obvykle značíme malými písmeny z konce latinské abecedy $(\dots, x, y, z)$
- Řetězce, které mohou být složené z terminálních i neterminálních symbolů, jako např. větné formy, značíme malými řeckými písmeny $(\alpha, \beta, \gamma, \dots)$.
- Pravidla $\alpha \to \beta, \alpha \to \gamma$ se stejnou levou stranou často zapisujeme stručněji jako $\alpha \to \beta \mid \gamma$.

Uvedené konvence platí, pokud v textu není výslovně uvedeno jinak.

1.2.2 Chomského hierarchie gramatik a jazyků

Pojem gramatiky definoval lingvista Noam Chomsky a rozdělil gramatiky do čtyř skupin (typů) na základě různých omezení na tvar pravidel. Jeho práce (z konce 50. let) byla původně motivována úvahami o struktuře přirozeného jazyka, z dnešního pohledu má však především význam jako základní (a neobyčejně výhodné) rozdělení gramatik podle jejich popisné síly. *Chomského hierarchie* rozlišuje tyto čtyři (základní) typy gramatik:

typ 0 Libovolná gramatika je gramatikou typu 0; na tvar pravidel se nekladou žádné omezující požadavky. Někdy též se takové gramatiky označují jako gramatiky bez omezení či frázové gramatiky (phrase grammars).

typ 1 Gramatika je typu 1 (nebo též *kontextová*[1], Context-Sensitive, CSG, méně často též *monotónní*), jestliže pro každé její pravidlo $\alpha \to \beta$ platí $|\alpha| \leq |\beta|$ s eventuelní výjimkou pravidla $S \to \varepsilon$, pokud se S nevyskytuje na pravé straně žádného pravidla.

typ 2 Gramatika je typu 2 (též *bezkontextová*, Context-Free, CFG), jestliže každé její pravidlo je tvaru $A \to \alpha$, $|\alpha| \geq 0$.

typ 3 Gramatika je typu 3 (též *regulární* či *pravolineární*[2]), jestliže každé její pravidlo je tvaru $A \to aB$ nebo $A \to a$ s eventuelní výjimkou pravidla $S \to \varepsilon$, pokud se S nevyskytuje na pravé straně žádného pravidla. [3]

V dalším textu bude ukázáno (viz Věta 3.14), že ke každé gramatice typu 2 s pravidly $A \to \alpha$, kde $|\alpha| \geq 0$, lze sestrojit ekvivaletní gramatiku typu 2 splňující požadavek $|\alpha| \geq 1$ s eventuelní výjimkou pravidla $S \to \varepsilon$, kde S se nevyskytuje na pravé straně žádného pravidla. Z toho okamžitě plyne i analogické tvrzení pro typ 3, pokud bychom povolili pravidla tvaru $A \to \varepsilon$.

Řekneme, že jazyk L je regulární – typu 3 (resp. bezkontextový – typu 2, resp. kontextový – typu 1, typu 0) pokud existuje regulární (resp. bezkontextová, kontextová, typu 0) gramatika $\mathcal{G}$ taková, že $L(\mathcal{G}) = L$. Symbolem $\mathcal{L}_i,$ pro $i = 0, 1, 2, 3$ budeme značit třídu všech jazyků typu i.

Hierarchie gramatik také určuje příslušnou hierarchii *jazyků*.

Nyní je již zřejmý smysl "výjimky"týkající se pravidla $S \to \varepsilon$; kdybychom ji nepovolili, stal by se z libovolného (třeba i regulárního) jazyka po přidání prázdného slova jazyk typu 0, který nelze popsat ani kontextovou gramatikou. To by bylo značně nepřirozené – přidáním jediného slova se "charakter"obecně (a též i obvykle) nekonečného jazyka příliš nezmění.

Z definice je patrné, že každý nový typ gramatiky v Chomského hierarchii je specifikován zavedením dalších omezujících podmínek na typ předchozí – například každá regulární gramatika je také gramatikou bezkontextovou, kontextovou i typu 0. Naopak to ovšem neplatí. V této souvislosti se rovněž nabízí přirozená otázka, zda existuje jazyk,

1. Název "kontextová"je odvozen z toho, že uvedenou podmínku je možné ekvivalentně zformulovat také tak, že každé pravidlo je tvaru $\gamma A \delta \to \gamma \alpha \delta$, kde $|\alpha| \geq 1$ (tj. neterminál A se přepíše na α tehdy, je-li obklopen kontextem γ a δ). "Ekvivalentní formulací"myslíme to, že třída všech jazyků, které lze generovat kontextovými gramatikami, se nezmění.

2. Analogicky lze definovat gramatiky *levolineární* s pravidly tvaru $A \to Ba$ nebo $A \to a$. I tyto gramatiky se nazývají regulární.

3. U tohoto typu gramatik bývá někdy v uvedené definici povoleno $a \in \Sigma \cup \{\varepsilon\}$; třída jazyků generovaných těmito gramatikami se nezmění oproti standardnímu případu.

který není typu 0, tj. jazyk, který nelze generovat žádnou gramatikou, či ekvivaletně jazyk, pro nějž neexistuje konečná reprezentace (pomocí gramatiky). Odpověď podává tato věta:

Tvrzení 1.4. *Nad abecedou $\{a\}$ existuje jazyk, který není typu 0.*

Důkaz. Množina všech slov nad abecedou $\{a\}$ je spočetně nekonečná. Množina všech jazyků nad touto abecedou má proto mohutnost $2^{\aleph_0}$, což je mohutnost kontinua (zejména je tedy nespočetná). Ukážeme, že jazyků typu 0 nad abecedou $\{a\}$ je pouze spočetně mnoho.

Buď M libovolná, ale pro další úvahy pevně zvolená spočetná množina. Ke každé gramatice $\mathcal{G}$ s množinou neterminálů N existuje ekvivalentní gramatika $\mathcal{G}'$ jejíž množina neterminálů je podmnožinou M (stačí přejmenovat prvky N vhodně zvolenými prvky množiny M). Lze proto bez újmy na obecnosti předpokládat, že každá gramatika s množinou terminálů $\{a\}$ má neterminály z množiny M. Ukážeme, že všech takových gramatik je pouze spočetně mnoho. K tomu si stačí uvědomit, že zápis každé takové gramatiky je vlastně slovo nad spočetnou abecedou

$$M \cup \{a, \rightarrow, \underline{(}, \underline{)}, \underline{\{}, \underline{\}}, \underline{,}\}$$

Podtržítka mají jen pomocnou úlohu – naznačují, že se má podtržený znak chápat jako prvek množiny a ne jako metasymbol.

Všech slov délky i nad touto abecedou je $\aleph_0^i = \aleph_0$ pro libovolné $i \in \mathbb{N}$. *Všech* slov nad touto abecedou je proto spočetně mnoho, neboť sjednocením spočetně mnoha spočetných množin je spočetná množina. Uvažovaných gramatik je proto rovněž spočetně mnoho. □

Z důkazu předchozí věty je patrné, že gramatikami lze ve skutečnosti popsat jen "velmi malou"třídu jazyků. Z uvedeného není ovšem jasné, jakého "druhu"jsou jazyky, které nejsou typu 0 – jak uvidíme později, spadají sem i některé velmi přirozeně definované jazyky.

Pozorného čtenáře jistě napadne i další otázka – existuje nějaký mocnější aparát pro popis jazyků než gramatiky? Uvědomme si, že základní požadavek na každý takový aparát je, aby jazyky byly popsány *konečným způsobem.* Jazyky tedy budou v každém případě specifikovány *konečnou* posloupností matematických symbolů. Jelikož matematika vystačí s konečně mnoha symboly, lze aplikovat argument předchozího důkazu; *každá* konečná reprezentace (popisný aparát) proto dokáže popsat nejvýše spočetnou množinu jazyků. Toto základní omezení nelze překonat. Není však možné předem vyloučit existenci aparátu, který interpretuje zápis jazyka takovým způsobem, že lze konečně zapsat i jazyky, které nejsou typu 0.

Toto je základní omezení, kterým je ovlivněna celá informatika. I program (nebo libovolný jiný zápis algoritmu) je totiž konečná posloupnost znaků a je jich proto nejvýše spočetně mnoho. Jak uvidíme, lze *problémy* formálně specifikovat jako jazyky. Všech problémů je tedy nespočetně mnoho. Z toho okamžitě plyne, že jsou i takové problémy, na jejichž řešení neexistuje algoritmus. Mezi nimi se najdou i takové problémy, kde by existence algoritmu byla velmi užitečná (ale bohužel tomu tak není). Patří k nim například

- problém, zda libovolný daný program ukončí svůj výpočet pro (jeden) libovolný daný vstup (vstupní data), resp. pro každý daný vstup (tj. zda program pro zadaný bude cyklit, resp. pro žádný vstup nikdy cyklit nebude),

- problém, zda dva libovolné dané programy jsou ekvivalentní v tom smyslu, že pro stejné vstupy dávají tytéž výsledky (například prototyp nějakého systému či jeho proveditelná specifikace jako jeden program a efektivní implementace téhož systému jako program druhý),
- problém, zda dvě libovolné bezkontextové gramatiky (tj. užitečný nástroj pro specifikaci syntaxe programovacích jazyků) jsou ekvivaletní (jedna z gramatik definuje syntaxi jazyka vhodnou pro uživatele – programátora, je však nevhodná pro jeho implementaci; tvůrce překladače musí najít jinou vhodnou gramatiku, ale neexistuje algoritmus, který by ověřil, zda tyto gramatiky jsou ekvivalentní)
- a řada dalších.

Touto problematikou se budeme zabývat v kapitole 5, tj. až po studiu jazyků generovatelných gramatikami v Chomského hierarchii. Detailněji se těmito otázkami zabývá teorie vyčíslitelnosti.

Kapitola 2

Regulární jazyky a konečné automaty

V této kapitole se budeme zabývat vlastnostmi regulárních jazyků. Ukážeme si alternativní způsoby jejich formální reprezentace, které jsou na první pohled velmi odlišné od regulárních gramatik – nejprve zavedeme pojem konečného automatu, který je matematickým modelem jednoduchého výpočetního zařízení s konečnou pamětí, schopného rozpoznávat určitý jazyk. Dokážeme, že třída jazyků, které lze rozpoznat konečnými automaty, je přesně třída regulárních jazyků; tento poznatek také přinese zajímavé výsledky o jejich struktuře a vlastnostech.

Další způsob formální reprezentace regulárních jazyků představují tzv. regulární výrazy, kterými se budeme rovněž zabývat. Umožňují popsat libovolný regulární jazyk jako výsledek kompozice několika jednoduchých operací nad jazyky (jde tedy o nerekurzivní popis, na rozdíl od gramatik a konečných automatů).

V závěru kapitoly se také zmíníme o praktickém uplatnění prezentovaných teoretických poznatků; možnosti jsou velmi široké, a pečlivé studium tohoto textu proto rozhodně není ztrátou času.

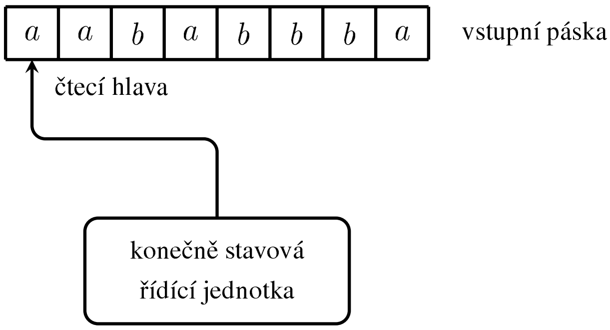
2.1 Konečné automaty

V každodenním životě se často setkáváme se zařízeními, která provádějí jistý druh činnosti na základě poměrně komplikované komunikace s okolím. Dobrým příkladem je třeba automat na kávu – představme si stroj, který je vybaven dvoumístným displejem (je tedy schopen přijmout hotovost až do výše 99 korun), dále otvorem pro mince, několika tlačítky pro výběr nápoje a samozřejmě výdejním systémem. Komunikace s automatem probíhá pomocí uvedených komponent. Jedinou výjimkou je v tomto směru displej; ten se komunikace přímo neúčastní, pouze signalizuje množství peněz, které byly do automatu vhozeny. To je také *jediná* veličina, která ovlivňuje další chování přístroje (pro jednoduchost neuvažujeme množství surovin, které v automatu zbývá). Určuje, které tlačítko pro volbu nápoje lze použít, zda je ještě možné vhodit další minci (pokud by výsledná částka přesáhla 99 korun, je mince odmítnuta), případně kolik peněz má automat po stisku speciálního tlačítka vrátit. Hodnota na displeji tedy přesně a úplně vystihuje momentální *stav* automatu. Ten se mění na základě komunikace s okolím podle předem stanoveného protokolu – vhozením další mince nebo stiskem tlačítka se stav automatu příslušným způsobem upraví. Odebrání nápoje přístroj nijak nezaznamená, nemá tudíž na stav žádný vliv (čtenář se může snadno přesvědčit, že tento předpoklad je celkem realistický). Vnitřní protokol automatu musí přesně vystihnout, která posloupnost akcí je pro automat přijatelná a která nikoliv. Jestliže bílá káva stojí 8 korun a objednává se tlačítkem X, je např. posloupnost akcí $2K\check{c}$, $5K\check{c}$, $1K\check{c}$, X pro automat přijatelná, zatímco sekvence $1K\check{c}$, $1K\check{c}$, X nikoliv. Jelikož další činnost automatu je *úplně* určena jeho momentálním stavem, kterých je *konečně mnoho* (přesně 100), je možné zmíněný protokol specifikovat tak, že pro každý stav uvedeme seznam akcí, na které je automat schopen reagovat spolu se stavem, do kterého se po dané akci dostane.

Existuje mnoho systémů, jejichž chování lze definovat pomocí konečně mnoha stavů, akcí a přechodů mezi stavy. Nemusí se vždy jednat zrovna o řídící jednotky – i některé společenské hry jako třeba šachy lze tímto způsobem chápat a přesně popsat; stavy jsou v tomto případě všechna možná rozložení figur na šachovnici (jelikož máme konečný počet figur i polí, je možných rozložení také konečně mnoho), akce jsou všechny možné tahy (např. „bílá věž z B2 na B6“) a v každém stavu lze provést pouze ty akce, které neodporují danému rozložení figur a pravidlům šachu. Pokud se zajímáme například o výherní strategii hráče s bílými figurami, můžeme stavy ve kterých dává bílý mat označit za *koncové*. Ověřit, zda bílý má v daném stavu šanci na výhru pak znamená zjistit, zda z daného stavu existuje posloupnost akcí, která vede do některého z koncových stavů. Ne každá hra se ovšem dá popsat jako systém s konečným počtem stavů (dále jen stručněji: *konečně stavový* systém). Příkladem jsou tzv. „piškvorky“ – hrací pole je potenciálně nekonečné a hra má tudíž nekonečně mnoho možných konfigurací.

I počítače jsou konečně stavové systémy, neboť mají sice velkou, ale přece jen konečnou paměť, která tudíž může nabýt pouze konečně mnoha stavů (počítáme sem samozřejmě i disky, vyrovnávací paměti, velkokapacitní záznamová média sdílená po síti a podobně). Akce a přechody mezi stavy paměti nelze v tomto případě popsat nějak jednoduše – závisí na konstrukci počítače a samozřejmě i na samotném obsazení paměti (jaký program se provádí, jaká má data atd.). Zároveň je však třeba poznamenat, že omezení na velikost paměti je poněkud umělé. V praxi nepředstavuje zásadní problém a v abstraktních úvahách je proto účelné tento limit zcela pominout. Získané teoretické výsledky pak mnohem lépe odvídají realitě, neboť přesněji vystihují „výpočetní sílu“ reálných počítačů, jak ostatně uvidíme v kapitole 5.

Abstraktním modelem konečně stavových systémů jsou tzv. *konečné automaty*. Konečný automat je vybaven konečně stavovou řídící jednotkou (tj. konečnou pamětí), čtecí hlavou a páskou, na které je zapsané vstupní slovo – viz obrázek 2.1. Na začátku výpočtu je hlava umístěna na nejlevějším políčku pásky. Automat na základě přečteného symbolu a momentálního stavu svůj stav změní a posune čtecí hlavu o jedno políčko vpravo. Výpočet končí, pokud se automat „zablokuje“, nebo přečte celé vstupní slovo. Slovo zapsané na pásce je automatem *akceptováno*, pokud je celé přečteno a výsledný stav je některý z



Obrázek 2.1: Konečný automat

předem určených *koncových* stavů. Množina všech slov, která daný konečný automat $\mathcal{M}$ akceptuje, tvoří *jazyk akceptovaný automatem* $\mathcal{M}$. Formální definice vypadá takto:

Definice 2.1. *Konečný automat* (Finite Automaton, FA) $\mathcal{M}$ je pětice $(Q, \Sigma, \delta, q_0, F)$, kde

- Q je neprázdná konečná množina *stavů*.
- Σ je konečná množina *vstupních symbolů*, nazývaná také *vstupní abeceda*.
- $\delta : Q \times \Sigma \to Q$ je parciální *přechodová funkce*.
- $q_0 \in Q$ je *počáteční stav*.
- $F \subseteq Q$ je množina *koncových stavů*.

Abychom mohli definovat jazyk přijímaný daným FA $\mathcal{M}$, zavedeme *rozšířenou* přechodovou funkci $\hat{\delta} : Q \times \Sigma^* \to Q$, definovanou induktivně vzhledem k délce slova ze Σ^*:

- $\hat{\delta}(q, \varepsilon) = q$ pro každý stav $q \in Q$.
- $\hat{\delta}(q, wa) = \begin{cases} \delta(\hat{\delta}(q, w), a) & \text{je-li } \hat{\delta}(q, w) \text{ i } \delta(\hat{\delta}(q, w), a) \text{ definováno,} \\ \perp & \text{jinak.} \end{cases}$

Symbol $\perp$ značí, že funkce není definovaná. Zápis $\hat{\delta}(q, w) = p$ znamená, že automat $\mathcal{M}$ přejde ze stavu q „pod slovem“ w (tj. postupným přečtením slova w znak po znaku zleva doprava) do stavu p. Jazyk *přijímaný (akceptovaný, rozpoznávaný)* konečným automatem $\mathcal{M}$, označovaný $L(\mathcal{M})$, je tvořen právě všemi takovými slovy, pod kterými automat přejde z počátečního stavu do některého z koncových stavů:

$$L(\mathcal{M}) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}$$

Jazyk, který je rozpoznatelný (nějakým) konečným automatem, nazveme *regulární* (viz však poznámka 2.2). *Ekvivalenci* konečných automatů definujeme podobně jako v případě gramatik – konečné automaty $\mathcal{M}$ a $\mathcal{M}'$ jsou ekvivalentní, pokud $L(\mathcal{M}) = L(\mathcal{M}')$.

Poznámka 2.2. *V části 1.2.2 jsme přívlastkem „regulární“ označili jazyk generovatelný regulární gramatikou; jak uvidíme, jsou třídy jazyků, které lze generovat regulárními gramatikami, resp. rozpoznat konečnými automaty, ve skutečnosti stejné. Než toto dokážeme (viz Věta 2.73), bude slovo „regulární“ jen zkratkou pro „rozpoznatelný konečným automatem“.*

Příklad 2.3. *Nechť* $\mathcal{M} = (\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_2\})$ *je FA, kde*

$$\begin{array}{ll} \delta(q_0, a) = q_1 & \delta(q_0, b) = q_2 \\ \delta(q_1, a) = q_2 & \delta(q_1, b) = q_0 \\ \delta(q_2, a) = q_0 & \delta(q_2, b) = q_1 \end{array}$$

Pak $L(\mathcal{M}) = \{w \in \{a, b\}^* \mid (\#_a(w) - \#_b(w)) \bmod 3 = 2\}$.

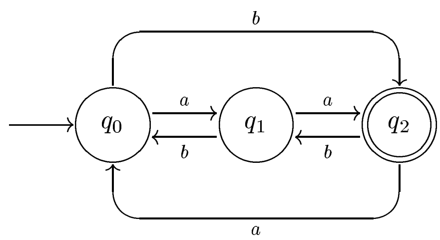
Úplná definice konkrétního automatu musí zahrnovat popis *všech* složek pětice z definice 2.1. Není však nutné tyto složky vždy reprezentovat standardní množinovou symbolikou (tj. výčtem prvků). V praxi se často používají i jiné (přehlednější) způsoby reprezentace konečných automatů. Předvedeme si tři z nich na automatu $\mathcal{M}$ z předchozího příkladu.

Automat $\mathcal{M}$ je možné reprezentovat pomocí *tabulky přechodové funkce* takto:

		a	b
$\rightarrow$	q_0	q_1	q_2
	q_1	q_2	q_0
$\leftarrow$	q_2	q_0	q_1

Stavy automatu jsou vypsány v záhlaví řádků, vstupní symboly v záhlaví sloupců, přechodová funkce je určena obsahem vnitřních polí tabulky (pokud je pro některé dvojice nedefinována, uvádí se v příslušném místě tabulky znak „–“), počáteční stav je označen znakem $\rightarrow$ a koncové stavy znakem $\leftarrow$.

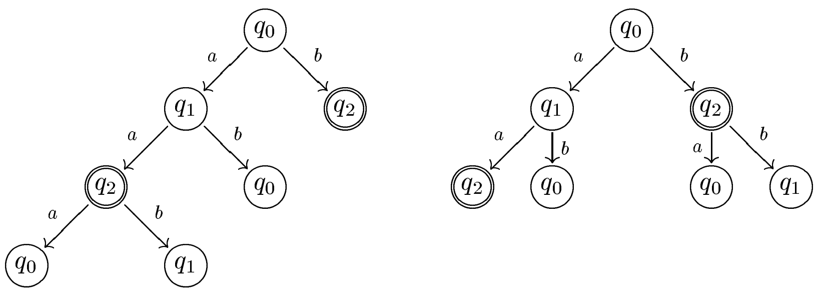
Ještě přehlednější, a proto nejčastěji používaná, je reprezentace pomocí *přechodového grafu* (též *přechodového systému s návěštími* ze Σ), který pro automat $\mathcal{M}$ vypadá takto:



Stavy odpovídají uzlům, přechodová funkce je znázorněna ohodnocenými hranami, vstupní abeceda je tvořena symboly, kterými jsou hrany ohodnoceny, počáteční stav je označen šipkou a koncové stavy jsou dvojitě zakroužkovány.

Nakonec ještě zmiňme reprezentaci výpočetním (či též stavovým) stromem. Kořen stromu odpovídá počátečnímu stavu (a není tedy nutné jej nějak označovat jako počáteční). Z každého uzlu, který není listem vychází – dle definice přechodové funkce – právě tolik hran ohodnocených symboly vstupní abecedy, kolik má odpovídající stav následníků (je-li tedy δ totální, pak právě tolik hran, kolik symbolů má vstupní abeceda). Jestliže nějaký stav odpovídá více uzlům, pak hrany vycházejí jen z jednoho z těchto uzlů. Výpočetním stromem lze reprezentovat jen ty automaty, kde každý stav je tzv. dosažitelný z počátečního stavu (viz definice 2.18) – z hlediska schopnosti rozpoznávat daný jazyk, není přítomnost či nepřítomnost nedosažitelných stavů podstatná (viz lemma 2.19).

Výpočetní strom pro daný automat není (obecně) určen jednoznačně – může se lišit dle toho, zda jej konstruujeme způsobem, který odpovídá například procházení do hloubky, nebo do šířky, či dalším možnostem. Pokud však ve výpočetním stromu ztotožníme uzly označené stejným stavem, obržíme přechodový graf (v němž však musíme vyznačit stav počáteční). Výpočetní strom pro automat $\mathcal{M}$ může tedy mít například tyto tvary:



Příklad 2.4. *Pro ilustraci nyní uvedeme* důkaz *faktu, že konečný automat $\mathcal{M}$ z příkladu 2.3 skutečně rozpoznává jazyk $L(\mathcal{M}) = \{w \in \{a,b\}^* \mid (\#_a(w) - \#_b(w)) \bmod 3 = 2\}$.*

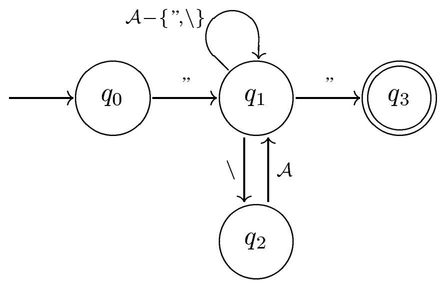
Důkaz. Dokážeme, že pro každé slovo $v \in \{a,b\}^*$ platí, že $\hat{\delta}(q_0, v) = q_i$, kde $i = (\#_a(v) - \#_b(v)) \bmod 3$. Indukcí k délce slova v:
$|\mathbf{v}| = \mathbf{0}$: Pak $v = \varepsilon$ a $\hat{\delta}(q_0, \varepsilon) = q_0$ podle definice rozšířené přechodové funkce. Zřejmě $(\#_a(\varepsilon) - \#_b(\varepsilon)) \bmod 3 = 0$.
Indukční krok: Nechť $v = ux$, kde $u \in \{a,b\}^*$ a $x \in \{a,b\}$. Podle indukčního předpokladu platí $\hat{\delta}(q_0, u) = q_i$, kde $i = (\#_a(u) - \#_b(u)) \bmod 3$. Nechť např. $x = a$ (případ kdy $x = b$ se vyšetří stejným způsobem). Přechodová funkce δ byla definována tak, že pro každé $k \in \{0,1,2\}$ platí $\delta(q_k, a) = q_l$, kde $l = (k+1) \bmod 3$. Proto také $\hat{\delta}(q_0, ua) = \delta(\hat{\delta}(q_0, u), a) = \delta(q_i, a) = q_j$, kde $j = (i+1) \bmod 3 = (\#_a(u) - \#_b(u) + 1) \bmod 3 = (\#_a(ua) - \#_b(ua)) \bmod 3$, což bylo dokázat.
Jelikož koncovým stavem je pouze q_2, platí $L(\mathcal{M}) = \{w \in \{a,b\}^* \mid \hat{\delta}(q_0, w) = q_2\} = \{w \in \{a,b\}^* \mid (\#_a(w) - \#_b(w)) \bmod 3 = 2\}$. □

Podobným způsobem lze postupovat i v jiných případech. Jediný problém (a tedy jádro důkazu) je postihnout vztah mezi slovy ze Σ^* a stavy daného automatu.

Přechodová funkce δ byla v definici 2.1 zavedena jako parciální, což umožňuje snadnější návrh a stručnější prezentaci konečných automatů – můžeme se soustředit jen na „důležité“ přechody z daného stavu.

Příklad 2.5. *Navrhneme konečný automat, rozpoznávající řetězcové konstanty podle (zjednodušené) konvence jazyka C – řetězec začíná uvozovkami, následuje posloupnost libovolných znaků s ASCII kódem 32–127 a na konci jsou zase uvozovky, před kterými ovšem nesmí být znak obráceného lomítka. Množinu znaků s ASCII kódem 32–127 označíme symbolem $\mathcal{A}$ (hrana s tímto návěštím $\mathcal{A}$ tedy reprezentuje, formálně vzato, množinu hran*

– pro každý z uvažovaných symbolů právě jedna hrana):



Parcialita přechodové funkce nemá podstatný vliv na výpočetní sílu konečných automatů, jak dokládá následující pomocné tvrzení:

Lemma 2.6. *Ke každému FA $\mathcal{M}$ existuje ekvivalentní FA $\mathcal{M}'$ s totální přechodovou funkcí.*

Důkaz. Nechť $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$. Automat $\mathcal{M}'$ sestrojíme tak, že ke stavům automatu $\mathcal{M}$ přidáme nový nekoncový stav p a chybějící přechody do něj „nasměrujeme". Tedy $\mathcal{M}' = (Q \cup \{p\}, \Sigma, \delta', q_0, F)$, kde $p \notin Q$ a δ' je definována pro všechna $q \in Q \cup \{p\}$ takto:

$$\delta'(q,a) = \begin{cases} \delta(q,a) & \text{je-li } \delta(q,a) \text{ definováno,} \\ p & \text{jinak.} \end{cases}$$

Zejména $\delta'(p,a) = p$ pro každé $a \in \Sigma$. Indukcí k délce slova se snadno ověří, že pro každé $q \in Q$ a $w \in \Sigma^*$ platí

$$\hat{\delta}'(q,w) = \begin{cases} \hat{\delta}(q,w) & \text{je-li } \hat{\delta}(q,w) \text{ definováno,} \\ p & \text{jinak.} \end{cases}$$

Jelikož $p \notin F$, platí $L(\mathcal{M}) = L(\mathcal{M}')$. □

Jazyk akceptovaný konečným automatem je možné definovat také pomocí pojmů *konfigurace* a *krok výpočtu*. Jak uvidíme, tento způsob je obecnější – lze ho aplikovat i na složitější modely, než jakými jsou konečné automaty. Proto tuto alternativu rovněž uvedeme.

Konfigurace konečného automatu $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ je každá dvojice $(q, w) \in Q \times \Sigma^*$. Na množině všech konfigurací automatu $\mathcal{M}$ zavedeme binární relaci *krok výpočtu*, označovanou $\vdash$, pomocí předpisu

$$(q, aw) \ \vdash \ (p, w) \ \overset{def}{\Longleftrightarrow} \ \delta(q,a) = p$$

Reflexivní a tranzitivní uzávěr relace kroku výpočtu značíme $\vdash^*$. Jazyk akceptovaný automatem $\mathcal{M}$ pak můžeme definovat také takto:

$$L(\mathcal{M}) = \{w \in \Sigma^* \mid (q_0, w) \vdash^* (q, \varepsilon),\ \text{kde } q \in F\}$$

Konfigurace konečného automatu $\mathcal{M}$ přesně popisuje momentální stav výpočtu, který $\mathcal{M}$ na daném slově provádí. Obsahuje úplnou informaci, která je potřebná pro jeho další pokračování – první komponenta konfigurace (tj.stav) reprezentuje informaci o dosud přečtené části vstupu, druhá komponenta udává dosud nezpracovanou část vstupu. „Programem“ pro tento výpočet je samozřejmě přechodová funkce.

Důkaz faktu, že obě uvedené definice jazyka $L(\mathcal{M})$ jsou ekvivalentní, lze přenechat čtenáři jako jednoduché cvičení – stačí ukázat platnost ekvivalence

$$\hat{\delta}(q, w) = p \iff (q, w) \vdash^* (p, \varepsilon) ,$$

což se dá jednoduše provést indukcí vzhledem k délce slova w.

2.1.1 Konstrukce konečných automatů

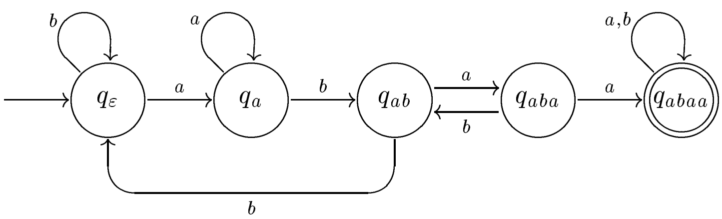
Konstrukce konečného automatu, který rozpoznává daný jazyk, je obecně netriviální úkol. V této části si ukážeme některé metody, s jejichž pomocí je možné vyřešit řadu konkrétních úloh.

Základním trikem, který dokáže zjednodušit návrh konečného automatu, je zavedení jisté pomocné struktury na stavech. Uvědomme si, že stavy konečného automatu představují konečnou paměť, do níž je možné ukládat informace o dosud přečtené části vstupního slova. Informaci, která je spojená s daným stavem, je účelné zachytit v jeho označení.

Příklad 2.7. *Máme za úkol sestrojit automat rozpoznávající jazyk*

$$L = \{w \in \{a, b\}^* \mid w \text{ obsahuje podslovo } abaa\}$$

Označení stavů automatu zvolíme tak, aby bylo patrné, jaká část požadovaného podslova $abaa$ *již byla automatem přečtena:*



Vhodná volba množniny stavů („struktury na stavech“) dokáže konstrukci automatu zjednodušit a výsledný automat zpřehlednit. V některých případech je její zavedení dokonce nevyhnutelné, má-li být definice technicky zvládnutelná.

Příklad 2.8. *Sestrojme automat rozpoznávající jazyk*

$$L = \{w \in \{a, b\}^* \mid \#_a(w) \bmod 1997 = 483 \ \wedge\ \#_b(w) \bmod 1998 = 645\}$$

Ve stavech automatu je třeba zachytit informaci o počtu dosud přečtených symbolů „a“ modulo 1997 *a o počtu dosud přečtených symbolů „b“ modulo* 1998. *Celkem tedy bude zapotřebí* $1997.1998 = 3990006$ *stavů. Reprezentovat takovýto automat pomocí přechodového grafu není nijak přehledné (i když stále možné). Místo toho zavedeme jednoduchou strukturu na stavech, která umožní zapsat celou definici na několik (krátkých) řádků. Nechť* $\mathcal{M} = (Q, \{a, b\}, \delta, q_{0,0}, \{q_{483,645}\})$, *kde*

$$Q = \{q_{i,j} \mid 0 \le i \le 1996 \ \wedge\ 0 \le j \le 1997\}$$

a přechodová funkce δ je definována takto:

- $\delta(q_{i,j}, a) = q_{i+1,j}$ *pro každé* $0 \leq i \leq 1995, 0 \leq j \leq 1997$
- $\delta(q_{1996,j}, a) = q_{0,j}$ *pro každé* $0 \leq j \leq 1997$
- $\delta(q_{i,j}, b) = q_{i,j+1}$ *pro každé* $0 \leq i \leq 1996, 0 \leq j \leq 1996$
- $\delta(q_{i,1997}, b) = q_{i,0}$ *pro každé* $0 \leq i \leq 1996$

Další často používanou technikou je *synchronní paralelní kompozice* automatů. Pro dané automaty $\mathcal{M}_1$ a $\mathcal{M}_2$ umožňuje snadno sestrojit automat rozpoznávající průnik (případně také sjednocení nebo rozdíl) jazyků $L(\mathcal{M}_1)$ a $L(\mathcal{M}_2)$. Intuitivně si lze celou konstrukci představit tak, že automaty $\mathcal{M}_1$ a $\mathcal{M}_2$ necháme běžet paralelně na stejném vstupním slově. Jejich běh je synchronní, tj. $\mathcal{M}_1$ a $\mathcal{M}_2$ provádějí kroky výpočtu vždy ve stejném okamžiku. Má-li slovo w patřit do sjednocení $L(\mathcal{M}_1)$ a $L(\mathcal{M}_2)$, musí být alespoň jeden z automatů po zpracování slova w v koncovém stavu. Formálně je tato myšlenka zachycena v níže uvedené definici.

Definice 2.9. Pro dané FA $\mathcal{M}_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, $\mathcal{M}_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, jejichž přechodové funkce jsou totální, definujeme konečný automat $\mathcal{M}_1 \Cup \mathcal{M}_2 = (Q_1 \times Q_2, \Sigma, \delta, (q_1, q_2), F)$, kde

- $F = \{(p, q) \mid p \in F_1 \vee q \in F_2\} = (F_1 \times Q_2) \cup (Q_1 \times F_2)$
- $\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$.

Předpoklad, že přechodové funkce δ_1, δ_2 jsou totální sice není omezující (viz lemma 2.6), avšak pro „správné“ fungovaní synchronní paralelní kompozice $\mathcal{M}_1 \Cup \mathcal{M}_2$ nezbytný; není pak totiž možné, aby se jedna z komponent na daném slově „zablokovala“, zatímco druhá měla možnost ve výpočtu pokračovat.

Věta 2.10. *Nechť* $\mathcal{M}_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ *a* $\mathcal{M}_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ *jsou konečné automaty s totálními přechodovými funkcemi. Pak* $L(\mathcal{M}_1 \Cup \mathcal{M}_2) = L(\mathcal{M}_1) \cup L(\mathcal{M}_2)$.

Důkaz. Nejprve dokážeme toto tvrzení:

$$\hat{\delta}((q_1, q_2), w) = (p, q) \iff \hat{\delta}_1(q_1, w) = p \wedge \hat{\delta}_2(q_2, w) = q \tag{2.1}$$

Důkaz se provede indukcí vzhledem k $|w|$.

- $|\mathbf{w}| = \mathbf{0}$: Platí $\hat{\delta}((q_1, q_2), \varepsilon) = (q_1, q_2)$, $\hat{\delta}_1(q_1, \varepsilon) = q_1$, $\hat{\delta}_2(q_2, \varepsilon) = q_2$. Pro $w = \varepsilon$ tedy obě strany ekvivalence, kterou je třeba dokázat, platí (přímo z definice rozšířené přechodové funkce). Samotná ekvivalence je proto rovněž platná.
- **Indukční krok:** Nechť $w = va$, kde $v \in \Sigma^*$, $a \in \Sigma$. Platí $\hat{\delta}((q_1, q_2), va) = (p, q) \iff$ existují stavy r, s takové, že $\hat{\delta}((q_1, q_2), v) = (r, s) \wedge \delta((r, s), a) = (p, q)$ $\iff \hat{\delta}_1(q_1, v) = r \wedge \hat{\delta}_2(q_2, v) = s$ (indukční předpoklad) $\wedge\ \delta_1(r, a) = p \wedge \delta_2(s, a) = q$ (dle definice δ) $\iff \hat{\delta}_1(q_1, va) = p \wedge \hat{\delta}_2(q_2, va) = q$.

Nyní již lze snadno dokázat vlastní tvrzení věty: $w \in L(\mathcal{M}_1 \Cup \mathcal{M}_2) \iff \hat{\delta}((q_1, q_2), w) = (p, q)$, kde $p \in F_1$ nebo $q \in F_2$ $\iff$ $\hat{\delta}_1(q_1, w) = p \wedge \hat{\delta}_2(q_2, w) = q$, kde $p \in F_1$ nebo $q \in F_2$ $\iff$ $w \in L(\mathcal{M}_1) \cup L(\mathcal{M}_2)$. □

Poznámka 2.11. *Podobným způsobem lze pro automaty* $\mathcal{M}_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ *a* $\mathcal{M}_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ *s totálními přechodovými funkcemi sestrojit automat* $\mathcal{M}_1 \Cap \mathcal{M}_2$, *resp.* $\mathcal{M}_1 \ominus \mathcal{M}_2$, *rozpoznávající jazyk* $L(\mathcal{M}_1) \cap L(\mathcal{M}_2)$, *resp.* $L(\mathcal{M}_1) - L(\mathcal{M}_2)$. *Jediný rozdíl je v definici množiny koncových stavů; ta je v případě* $\mathcal{M}_1 \Cap \mathcal{M}_2$ *rovna* $F_1 \times F_2$ *a v případě* $\mathcal{M}_1 \ominus \mathcal{M}_2$ *je definována jako* $\{(p,q) \mid p \in F_1 \wedge q \notin F_2\} = F_1 \times Q_2 - Q_1 \times F_2$. *Důkazy, že* $L(\mathcal{M}_1 \Cap \mathcal{M}_2) = L(\mathcal{M}_1) \cap L(\mathcal{M}_2)$ *a* $L(\mathcal{M}_1 \ominus \mathcal{M}_2) = L(\mathcal{M}_1) - L(\mathcal{M}_2)$ *jsou snadné; použije se vztah (2.1).*

Pro úplnost ještě poznamenejme, že pro automat $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ *s totální přechodovou funkcí lze také lehce sestrojit automat* $\overline{\mathcal{M}}$ *rozpoznávající jazyk* co–$L(\mathcal{M})$ *– stačí položit* $\overline{\mathcal{M}} = (Q, \Sigma, \delta, q_0, Q - F)$. *Zřejmě* $w \in L(\overline{\mathcal{M}}) \iff w \notin L(\mathcal{M})$, *tedy* $L(\overline{\mathcal{M}}) = \Sigma^* - L(\mathcal{M})$ *(předpoklad, že* δ *je totální, je opět zcela nezbytný).*

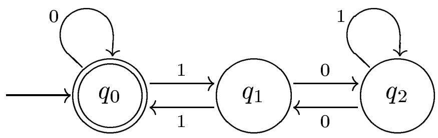
Příklad 2.12. *Nechť* $L \subseteq \{0,1\}^*$ *je jazyk, obsahující všechna slova* w, *která vyhovují těmto podmínkám:*

1. w *je binární zápis čísla dělitelného třemi (*ε *chápeme jako jiný zápis čísla* 0*) a*
2. w *obsahuje lichý počet výskytů znaku „*0*“*

Prvky L *jsou například slova* $000, 011, 1011010$. *Konečný automat rozpoznávající jazyk* L *sestrojíme jako paralelní kompozici dvou jednodušších automatů, které rozpoznávají slova vyhovující podmínce 1 resp. 2. Automat* $\mathcal{M}_1$ *rozpoznávající jazyk*

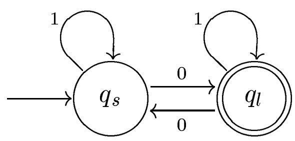
$$L_1 = \{w \in \{0,1\}^* \mid w \text{ je binární zápis čísla dělitelného třemi}\}$$

vypadá takto:

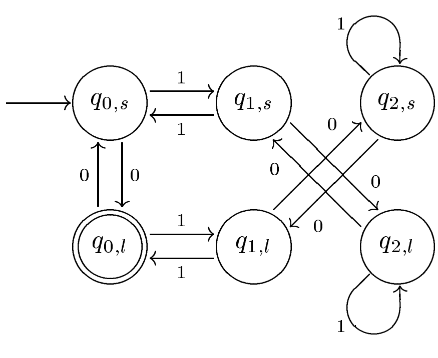


Indexy stavů odpovídají zbytkovým třídám modulo 3 *(je třeba si uvědomit, že připsáním znaku „0“ na konec binárního čísla se zdvojnásobí jeho hodnota; připsáním „1“ dosáhneme zdvojnásobení a přičtení jedničky. V obou případech se snadno zjistí, jak se změní zbytek při dělení třemi.)*

Automat $\mathcal{M}_2$ *rozpoznávající jazyk* $L_2 = \{w \in \{0,1\}^* \mid \#_0(w) \bmod 2 = 1\}$ *je jednoduchý:*



Výsledný automat $\mathcal{M}_1 \Cap \mathcal{M}_2$, *kdy namísto dvojice* (q_i, q_j) *píšeme* $q_{i,j}$, *vypadá takto:*



Výčet metod a triků, které lze při návrhu konečných automatů použít, není zdaleka úplný. V části 2.2 se seznámíme s některými rozšířeními základního modelu konečných automatů, které sice nemají vliv na výpočetní sílu (ukážeme, že tyto „obecnější“ stroje lze ve skutečnosti vždy simulovat konečným automatem), avšak jejich konstrukce je často velmi přímočará. Otevírá se tak další strategie pro návrh konečných automatů – nejprve navrhneme „rozšířený“ stroj a ten pak transformujeme na ekvivalentní konečný automat.

2.1.2 Lemma o vkládání pro regulární jazyky

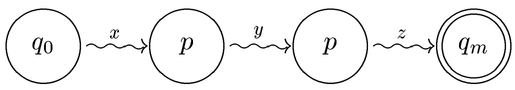
O tom, že nějaký jazyk je regulární, se můžeme (více či méně) snadno přesvědčit konstrukcí příslušného automatu. V případě, že se nám takový automat zkonstruovat nepodaří, může to mimo jiné znamenat, že *neexistuje*. Jak to však dokážeme? Základními nástroji, které pro tento účel mohou posloužit, jsou tzv. lemma o vkládání (též známé jako „pumping lemma“) pro regulární jazyky, které je nutnou (nikoli však postačující) podmínkou pro regularitu jazyka, a tzv. Myhillova-Nerodova věta (viz odstavec 2.1.3), která představuje podmínku nutnou a postačující.

Lemma 2.13 (o vkládání). *Nechť L je regulární jazyk. Pak existuje $n \in \mathbb{N}$ takové, že libovolné slovo $w \in L$, jehož délka je alespoň n, lze psát ve tvaru $w = xyz$, kde $|xy| \leq n$, $y \neq \varepsilon$ a $xy^iz \in L$ pro každé $i \in \mathbb{N}_0$. (Číslo n se neformálně nazývá pumpovací konstanta.)*

Důkaz. Jelikož L je regulární, existuje deterministický FA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ rozpoznávající jazyk L. Položme $n = card(Q)$. Ukažme, že pro libovolné slovo $w \in L$ délky alespoň n (tj. $w = a_1 \ldots a_m, m \geq n$) platí, že automat $\mathcal{M}$ projde při akceptování slova w (alespoň) dvakrát stejným stavem: $\mathcal{M}$ provede výpočet

$$(q_0, a_1 \ldots a_m) \vdash (q_1, a_2 \ldots a_m) \vdash \ldots \vdash (q_m, \varepsilon), \text{ kde } q_m \in F$$

při němž projde $m + 1$, tj. více než n konfiguracemi. Podle Dirichletova principu[1] se tedy alespoň dvě z konfigurací musí shodovat ve svých prvních komponentách – stavech (těch je jen n). Jinak řečeno, existují indexy i, j takové, že $0 \leq i < j \leq n$ a $q_i = q_j = p$. Schématicky:



Slovo w se tedy rozpadne na tři části – $w = xyz$, kde $x = a_1 \ldots a_i, y = a_{i+1} \ldots a_j, z = a_{j+1} \ldots a_m$ a kde $y \neq \varepsilon$. Jinak řečeno $\hat{\delta}(q_0, x) = p$, $\hat{\delta}(p, y) = p$ a $\hat{\delta}(p, z) = q_m$. Je zřejmé, že ke zopakování nějakého stavu dojde nejpozději po zpracování prvních n znaků[2] slova w, a tedy dostáváme $|xy| \leq n$. Dále $\hat{\delta}(p, y^i) = p$ pro libovolné $i \in \mathbb{N}_0$, proto také $\hat{\delta}(q_0, xy^iz) = q_m$, tj. $xy^iz \in L(\mathcal{M})$ pro každé $i \in \mathbb{N}_0$. □

---

1. Nechť P, Q jsou konečné množiny takové, že $card(P) > card(Q)$. Pak libovolná funkce $f : P \to Q$ není prostá.
2. bez uvážení tohoto faktu bychom obrželi o něco slabší variantu lemmatu: Nechť L je regulární jazyk. Pak existuje $n \in \mathbb{N}$ takové, že libovolné slovo $w \in L$, jehož délka je alespoň n, lze psát ve tvaru $w = xyz$, kde $1 \leq |y| \leq n$ a $xy^iz \in L$ pro každé $i \in \mathbb{N}_0$.

Je užitečné si uvědomit, že PL (díky alternování universálních a existenčních kvantifikátorů) lze zapsat takto :

$$\begin{aligned}
&(\text{Nechť})\ L \text{ je regulární} \implies \quad \exists n \in \mathbb{N}.\\
&\qquad \forall w \in L.\ |w| \geq n\,.\\
&\qquad\quad \exists\, x, y, z.\ \ w = xyz \wedge\\
&\qquad\qquad y \neq \varepsilon \wedge\\
&\qquad\qquad |xy| \leq n\,.\\
&\qquad\quad \forall i \geq 0.\ xy^i z \in L
\end{aligned}$$

Zdůrazněme, že lemma o vkládání (na rozdíl od Myhillovy-Nerodovy věty, kterou zformulujeme a dokážeme v následujícím odstavci – viz 2.1.3) poskytuje podmínku, která je pouze *nutná*, ale nikoliv *postačující* pro to, aby daný jazyk byl regulární. Lze tedy pomocí něho dokázat, že nějaký jazyk regulární *není* (tím, že prokážeme jeho nesplnění), ale v žádném případě ne to, že regulární *je*.

Pumping lemma (PL) je tvrzení tvaru implikace $\underline{L \text{ je regulární} \implies Q}$. Při dokazování, že L není regulární použijeme kontrapositivní formu PL, tj. $\underline{\neg Q \implies L \text{ není regulární}}$, či ekvivaletně důkaz sporem: L je regulární $\implies Q \wedge \neg Q$. V každém případě jde však o dokázání $\neg Q$. Obecně tedy můžeme postupovat takto: (pro dosažení sporu s PL předpokládejme, že L *je* regulární; pak musí splňovat podmínky pumping lemmatu a ukážeme, že tomu tak není) tedy ukážeme platnost $\neg Q$, tj. že

- pro *libovolné* $n \in \mathbb{N}$ (pumpovací konstantu)
- *existuje* takové slovo $w \in L$, které má délku alespoň n, a pro které platí, že
- při *libovolném* rozdělení slova w na takové tři části x, y, z, že $|xy| \leq n$ a $y \neq \varepsilon$,
- *existuje* (alespoň jedno) $i \in \mathbb{N}_0$ takové, že $xy^i z \notin L$.

Pak z PL plyne, že L není regulární.

Znovu si tedy uvědomme, že při použití PL k důkazu, že jazyk není regulární, volíme slovo w a počet pumpování i (viz výše podtržené existenční kvantifikátory). Nevolíme ani pumpovací konstantu n, ani rozdělení na podslova x, y, z.

Příklad 2.14. *Ukážeme, že* $L = \{a^p \mid p \text{ je prvočíslo}\}$ *nad abecedou* $\{a\}$ *není regulární.*

Důkaz. Pro dosažení sporu předpokládejme, že L *je* regulární. Buď $n \in N$ libovolné (pumpovací konstanta z PL). Jelikož prvočísel je nekonečně mnoho, existuje prvočíslo p, které je větší nebo rovno n; zvolme $w = a^p$ patřící do L. Při jakémkoli rozdělení w na podslova x, y, z musí být $y = a^k, k \geq 1$. Napumpujeme-li y $p + 1$-krát, dostaneme: $xy^{p+1}z = xyy^pz = xyzy^p = a^pa^{kp} = a^{p(k+1)}$, což je jistě slovo, které nepatří do jazyka L, protože $p(k+1)$ není prvočíslo – dostáváme tedy spor s naším předpokladem, že L je regulární. Podle PL tedy L regulární není. □

Příklad 2.15. *Jazyk* $L = \{a^ib^i \mid i \in \mathbb{N}\}$ *nad abecedou* $\{a, b\}$ *není regulární.*

Důkaz. Nyní již poněkud stručněji: buď $n \in \mathbb{N}$ libovolné. Slovo a^nb^n jistě patří do L; pokud ho jakkoli rozdělíme na tři části x, y, z tak, že $|xy| \leq n$ a $|y| \geq 1$, pak x a y jsou složeny pouze ze symbolu a, tj. $x = a^k$, $y = a^l$, $z = a^{n-k-l}b^n$, kde $1 \leq l, k+l \leq n$. Pak pro $i = 2$ dostáváme $a^ka^{2\cdot l}a^{n-k-l}b^n \notin L$, neboť $k+2l+n-k-l = n+l \neq n$. Obdobně bychom ke sporu dospěli volbou $i = 0$ (volba $i = 1$ by byla jen naše „nedostatečnost“). □

Formule s alternujícími kvantifikátory a hra 2 hráčů. Pro porozumění formuli s větším počtem kvantifikátorů je možné na ni nahlížet jako na hru dvou hráčů , kde universálnímu kvantifikátoru $\forall$ odpovídá hráč Al a existeční kvantifikátor $\exists$ je reprezentován hráčem Ex. Al a Ex hrají proti sobě (jsou na tahu) v tom pořadí, které odpovídá výskytu kvantifikátorů ve formuli (čteno zleva doprava).

Je-li na tahu Al, snaží se formuli tvaru $\forall u.p(u)$ vyvrátit: je-li $\forall u.p(u)$ pravda, pak Al nemůže vyhrát; je-li $\forall u.p(u)$ nepravda, pak existuje aspoň jedna hodnota u, která vyvrací $p(u)$ a Al může vyhrát tak, že pro u zvolí právě tuto hodnotu.

Je-li na tahu Ex, snaží se formuli tvaru $\exists u.p(u)$ učinit pravdivou: je-li $\exists u.p(u)$ pravda, Ex může vyhrát volbou hodnoty pro u takovou, že $p(u)$ je pravda. Je-li $\exists u.p(u)$ nepravda, nemůže se mu toto podařit a hru prohrává.

Vraťme se nyní zpět k pumping lemmatu; to tvrdí, že pokud L je regulární jazyk, pak

1. $\exists n \in \mathbb{N}$
2. $\forall w \in L$ takové, že $|w| \geq n$
3. $\exists\, x, y, z$ taková, že $w = xyz \ \wedge \ y \neq \varepsilon \ \wedge \ |xy| \leq n$
4. $\forall i \geq 0 : \ xy^i z \in L$

Toto tvrzení obsahuje 4 kvantifikátory a podmínky za nimi uvedené budou ve hře 2 hráčů přestavovat omezení na možnosti volby, které každý z hráčů bude během hry činit. Zvolme nějaký regulární jazyk L; hra pro L probíhá takto:

1. Ex zvolí přirozené číslo n,
2. Al zvolí w, a to tak, že $w \in L$ a $|w| \geq n$,
3. Ex zvolí x, y, z taková, že $w = xyz$ a $y \neq \varepsilon$ a $|xy| \leq n$
4. Al zvolí $i \geq 0$, přičemž se snaží volbu provést tak, aby vyhrál, tj. snaží se o $xy^i z \notin L$.

Demonstrovat vyhrávající strategii pro hráče Ex značí (de facto) znovu provést důkaz pumping lemmatu, tentokrát ovšem v pojmech hry: nechť tedy L je jazyk akceptovaný nějakým konečným automatem, řekněme $\mathcal{P}$.

1. Ex zvolí $n = card(Q)$, kde Q je množina stavů automatu $\mathcal{P}$.
2. Al zvolí slovo w takové, že $w \in L$ a $|w| \geq n$. (Pokud takové slovo neexistuje, pak Al prohrává: v tomto případě druhý kvantifikátor říká, že všechny prvky *prázdné* množiny mají jistou vlastnost, což je (bez ohledu na to o jakou vlastnost se jedná) triviálně pravda – Al nemůže formuli vyvrátit).
3. Nyní Ex najde v $\mathcal{P}$ akceptující výpočet pro w (ten jistě existuje, protože $w \in L$) a zaznamená si posloupnost p stavů řídicí jednotky, kterými se při akceptování w projde. Jelikož $|w| \geq n$, je těchto „průběžných" stavů alespoň $n + 1$, ale Q má jen n stavů, a tedy v p (akceptující posloupnosti stavů) se musí alespoň jeden ze stavů vyskytovat alespoň dvakrát (podle Dirichletova principu). Nechť q je první výskyt nějakého opakujícího se stavu v p. Ex rozdělí výpočet na 3 části (bude korespondovat postupnému přečtení řetězců x, y a z), a to podle prvních dvou výskytů konfigurací majících v 1. komponentě stav q. Výpočet pro vstupní slovo $w = xyz$ lze zapsat jako
$$(q_0, xyz) \vdash^* (q, yz) \vdash^+ (q, z) \vdash^* (q_f, \varepsilon),$$

kde x je přečteno dříve, než se poprvé vejde do stavu q, po následném přečtení y se (poprvé) vrátíme do q (tj. druhý výskyt q) a z je zbytkem vstupního slova. Jistě tedy platí, že x, y, z jsou taková, že $w = xyz$ a $y \neq \varepsilon$ a $|xy| \leq n$. Ex tedy splnil podmínky volby.

4. Ať Al nyní zvolí jakékoli $i \geq 0$, bude výpočet v $\mathcal{P}$ pro slovo xy^iz vždy tvaru

$$(q_0, xy^iz) \vdash^* (q, y^iz) \underbrace{\vdash^+ \ \ldots \ \vdash^+}_{i \text{ krát přečte } y} (q, z) \vdash^* (q_f, \varepsilon),$$

což je ale akceptující výpočet v $\mathcal{P}$, tj. $xy^iz \in L$, a tedy Al prohrává, Ex vítězí.

Zopakujme tedy, že použití pumping lemmatu k důkazu (sporem) neregularity nějakého jazyka L tedy v termínech hry probíhá takto:

1. Ex zvolí přirozené číslo n,
2. Al zvolí w, a to tak, že $w \in L$ a $|w| \geq n$,
3. Ex zvolí x, y, z taková, že $w = xyz$ a $y \neq \varepsilon$ a $|xy| \leq n$
4. Al zvolí $i \geq 0$, přičemž se snaží volbu provést tak, aby vyhrál, tj. snaží se o $xy^iz \notin L$.

Příklad 2.16. *Nechť L obsahuje právě všechna ta slova nad abecedou $\{a\}$, jejichž délky jsou druhými mocninami přirozených čísel, tj. $L = \{a^{n^2} \mid n \in \mathbb{N}\}$. Ukažme, že L není regulární, a to tak, že presentujeme vyhrávající strategii Al-a:*

1. *Ex zvolí přirozené číslo n.*
2. *Al zvolí $z \in L$ a $|z| > n^2$ (to vždy lze, protože L je nekonečný).*
3. *Ex zvolí u, v a w taková, že $z = uvw$ a $v \neq \varepsilon$ a $|uv| \leq n$.*
4. *Al zvolí $i = 2$, Protože $z \in L$, platí $|z| = m^2$ pro nějaké přirozené m. Al volil z tak, že platí $m > n$. Máme tedy $0 < |v| \leq n$ a označme $k = |v|$. Pak*

$$m^2 < m^2 + k = |uv^2w| \leq m^2 + n < m^2 + m < (m+1)^2 \ .$$

Délka slova uv^2w tedy padne mezi druhé mocniny dvou po sobě jdoucích přirozených čísel (m a $m+1$), takže $uv^2w \notin L$, a tedy Al vyhrává.

Jelikož Al má vyhrávající strategii, bez ohledu na to, jak Ex hraje, L nemůže být regulární.

Fakt, že lemma o vkládání neudává postačující podmínku pro regularitu jazyka, lze názorně demonstrovat tímto příkladem:

Příklad 2.17. *Jazyk $L = \{a, b\}^* \cup \{c^j a^i b^i \mid i, j \in \mathbb{N}\}$ nad abecedou $\{a, b, c\}$ splňuje podmínky lemmatu o vkládání, přitom však není regulární (jak lze snadno dokázat užitím Myhillovy-Nerodovy věty).*

2.1.3 Myhillova-Nerodova věta

V tomto odstavci zformulujeme a dokážeme velice důležité tvrzení, tzv. Myhillovu-Nerodovu větu, která představuje algebraickou charakterizaci třídy regulárních jazyků a má četné důležité důsledky. Abychom ji mohli zformulovat, potřebujeme několik pomocných pojmů.

Definice 2.18. Nechť $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ je konečný automat. Stav $q \in Q$ nazveme *dosažitelný*, pokud existuje $w \in \Sigma^*$ takové, že $\hat{\delta}(q_0, w) = q$. Stav je *nedosažitelný*, pokud není dosažitelný.

Je zřejmé, že vypustíme-li z daného automatu $\mathcal{M}$ nedosažitelné stavy, jazyk $L(\mathcal{M})$ se nezmění. Tuto transformaci lze navíc provést algoritmicky, jak dokládá toto lemma:

Lemma 2.19. *Existuje algoritmus, který pro každý konečný automat $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ sestrojí ekvivalentní konečný automat $\mathcal{M}'$ bez nedosažitelných stavů.*

Důkaz. Nejprve dokážeme, že množinu $Q' = \{q \in Q \mid q \text{ je dosažitelný}\}$ lze algoritmicky zkonstruovat. Uvědomme si, že do každého dosažitelného stavu vede v přechodovém grafu automatu $\mathcal{M}$ *konečná* cesta z počátečního stavu q_0. Označíme-li pro každé $i \in \mathbb{N}_0$ symbolem S_i množinu stavů, do kterých se lze z q_0 dostat cestou o délce nejvýše i (tj. použitím nejvýše i přechodů), platí:

$$Q' = \bigcup_{i \geq 0} S_i \tag{2.2}$$

Do stavu q_0 se vždy lze dostat cestou délky 0 – pro *každý* konečný automat tedy platí $S_0 = \{q_0\}$. Hodnoty S_i pro $i \geq 1$ již závisí na tom, jak je $\mathcal{M}$ definován. Můžeme je však snadno vypočítat podle následujícího induktivního předpisu:

- $S_0 = \{q_0\}$
- $S_{i+1} = S_i \cup \{q \mid \exists p \in S_i, a \in \Sigma : \delta(p, a) = q\}$

Indukcí vzhledem k i se snadno ověří, že každé S_i obsahuje pouze dosažitelné stavy. Dále pro každé $i \in \mathbb{N}_0$ platí, že $S_i \subseteq Q$ a $S_i \subseteq S_{i+1}$. Označme $n = \mathit{card}(Q)$. Vzhledem k tomu, že množina Q je konečná, nemohou se množiny S_i pro rostoucí i neustále zvětšovat – existuje tedy $k \leq n$ takové, že $S_k = S_{k+1}$. Z definice množin S_i nyní vyplývá, že dokonce pro každé $j \geq 0$ platí $S_k = S_{k+j}$. Proto můžeme množinu Q' všech dosažitelných stavů, tj. vztah 2.2 vyjádřit také jako

$$Q' = \bigcup_{i=0}^{k} S_i = S_k \tag{2.3}$$

Tato rovnost podává přesný návod na to, jak množinu Q' *vypočítat*. Formálně je tím dokázána správnost i konečnost algoritmu 2.1.

Hledaný automat $\mathcal{M}'$ je pak $(Q', \Sigma, \delta/Q', q_0, F \cap Q')$, kde symbol δ/Q' značí zobrazení δ zúžené na Q'. Přitom platí, že pokud je δ totální, je i δ/Q' totální. Fakt, že $L(\mathcal{M}) = L(\mathcal{M}')$, je zřejmý. $\square$

Způsob, jakým jsme v algoritmu 2.1 zkonstruovali množinu všech dosažitelných stavů konečného automatu $\mathcal{M}$, je velmi speciální aplikací Knasterovy-Tarského věty o pevném bodě a Kleeneovy věty o rekurzi. Tento princip použijeme ještě mnohokrát.

Definice 2.20. Nechť Σ je abeceda a nechť $\sim$ je ekvivalence na Σ^*. Řekneme, že $\sim$ je *pravá kongruence*, též *zprava invariantní ekvivalence*, pokud pro všechna $u, v, w \in \Sigma^*$ platí: $u \sim v \implies uw \sim vw$. *Index* ekvivalence $\sim$ je počet tříd rozkladu $\Sigma^*/_\sim$. Pokud je těchto tříd nekonečně mnoho, klademe index $\sim$ roven ∞.

Poznámka 2.21. *Snadno se nahlédne, že ekvivalence $\sim$ na Σ^* je pravá kongruence, právě když pro každé $u, v \in \Sigma^*, a \in \Sigma$ platí $u \sim v \implies ua \sim va$. (Z jedné strany triviální, obrácená implikace se snadno ukáže indukcí podle délky zprava přiřetězeného slova w.)*

Algoritmus 2.1 Eliminace nedosažitelných stavů konečného automatu.

Vstup: Konečný automat $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$.
Výstup: Ekvivalentní konečný automat $\mathcal{M}'$ bez nedosažitelných stavů.

$i := 0$; $S_0 := \{q_0\}$;
repeat
  $S_{i+1} := S_i \cup \{q \mid \exists p \in S_i, a \in \Sigma : \delta(p, a) = q\}$;
  $i := i + 1$;
until $S_i = S_{i-1}$;
$Q' := S_i$;
$\mathcal{M}' := (Q', \Sigma, \delta/Q', q_0, F \cap Q')$;

Konečné automaty a pravé kongruence s konečným indexem spolu velmi úzce souvisejí, jak ukazuje následující věta (a zejména její důkaz).

Věta 2.22. ***(Nerodova).*** *Nechť L je jazyk nad Σ. Pak tato dvě tvrzení jsou ekvivalentní:*

1. *L je rozpoznatelný konečným automatem.*
2. *L je sjednocením některých tříd rozkladu určeného pravou kongruencí na Σ^* s konečným indexem.*

Důkaz. $(\mathbf{1} \Longrightarrow \mathbf{2})$ Nechť $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ je FA, který rozpoznává L. Bez újmy na obecnosti předpokládejme, že $\mathcal{M}$ je bez nedosažitelných stavů (viz lemma 2.19) a δ je totální (viz lemma 2.6); pak také $\hat{\delta}$ je totální. Na Σ^* definujme binární relaci $\sim$ takto:

$$u \sim v \overset{def}{\Longleftrightarrow} \hat{\delta}(q_0, u) = \hat{\delta}(q_0, v)$$

Relace $\sim$ je relací ekvivalence, která sdružuje taková slova, pro která automat $\mathcal{M}$ přejde z q_0 do stejného stavu. Třídy rozkladu určeného relací $\sim$ tedy odpovídají stavům automatu $\mathcal{M}$, proto relace $\sim$ má konečný index. Ukažme, že $\sim$ je pravá kongruence. Nechť $u \sim v$ a $a \in \Sigma$. Pak $\hat{\delta}(q_0, ua) = \delta(\hat{\delta}(q_0, u), a) = \delta(\hat{\delta}(q_0, v), a) = \hat{\delta}(q_0, va)$, tedy $ua \sim va$, což bylo dokázat. Jazyk $L(\mathcal{M})$ je sjednocením těch tříd rozkladu určeného relací $\sim$, které odpovídají koncovým stavům automatu $\mathcal{M}$. Označíme-li symbolem $\langle q \rangle$ třídu, která odpovídá stavu q, platí $u \in L \iff \hat{\delta}(q_0, u) = q$, kde $q \in F \iff u \in \langle q \rangle$, kde $q \in F$.

$(\mathbf{2} \Longrightarrow \mathbf{1})$ Nechť L je sjednocením některých tříd rozkladu určeného pravou kongruencí $\sim$ na Σ^* s konečným indexem. Prvek faktorové množiny $\Sigma^*/_\sim$ (tj. třídu rozkladu) obsahující prvek u budeme značit $[u]$. Dále definujme konečný automat $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$, kde

- $Q = \Sigma^*/_\sim$, tj. stavy jsou třídy rozkladu na Σ^* určeného ekvivalencí $\sim$. Jelikož $\sim$ má konečný index, je těchto tříd konečně mnoho.
- δ je definována pomocí reprezentantů: $\delta([u], a) = [ua]$. Tato definice je korektní, tj. *nezávisí* na volbě konkrétního reprezentanta[3], neboť $\sim$ je zprava invariantní.
- $q_0 = [\varepsilon]$.

3. Nezávislost na volbě reprezentantů v tomto případě znamená, že pro každé $u, v \in \Sigma^*$, $a \in \Sigma$ platí $u \sim v \Longrightarrow ua \sim va$

- F obsahuje právě ty prvky $\Sigma^*/_\sim$, jejichž sjednocením obdržíme jazyk L.

Indukcí vzhledem k délce slova v se snadno ukáže, že $\hat{\delta}([\varepsilon], v) = [v]$ pro každé $v \in \Sigma^*$. Zřejmě $L = L(\mathcal{M})$, neboť $v \in L \iff [v] \in F \iff \hat{\delta}([\varepsilon], v) \in F$. □

Poznámka 2.23. *Každý konečný automat tedy jednoznačně určuje jistou pravou kongruenci s konečným indexem a obráceně. Omezíme-li se pouze na automaty, které jsou bez nedosažitelných stavů a s totální přechodovou funkcí, jsou obě uvedená přiřazení navzájem inverzní až na označení stavů automatu.*

Předchozí věta rovněž udává podmínku, která je nutná a postačující k tomu, aby daný jazyk byl regulární. Lze tedy pomocí ní dokázat i to, že nějaký jazyk regulární není.

Příklad 2.24. *Dokážeme, že jazyk $L = \{a^i b^i \mid i \in \mathbb{N}\}$ nad abecedou $\{a, b\}$ není rozpoznatelný žádným konečným automatem. (O tomto jazyku jsme již dokázali, že regulární není – viz příklad 2.15; tímto důkazem „jen“ ilustrujeme techniku obsaženou ve větě 2.22.)*

Důkaz. Předpokládejme, že L je regulární. Pak podle věty 2.22 existuje pravá kongruence $\sim$ na $\{a, b\}^*$ s konečným indexem taková, že L je sjednocením některých tříd rozkladu $\{a, b\}^*/_\sim$. Ukážeme, že to není možné; k tomu nám stačí nalézt dvě slova u, v, která prokazatelně leží ve stejné třídě rozkladu $\{a, b\}^*/_\sim$ a přitom $u \in L$ a $v \notin L$.

Buď k index ekvivalence $\sim$. Uvažme těchto $k + 1$ slov: $ab, aab, aaab, \ldots, a^{k+1}b$. Jelikož rozklad $\{a, b\}^*/_\sim$ má právě k tříd, musí ve výše uvedeném seznamu existovat dvě různá slova, která patří do *stejné* třídy – tedy $a^i b \sim a^j b$ pro nějaké $1 \leq i < j \leq k + 1$. Protože $\sim$ je zprava invariantní, platí rovněž $a^i b b^{i-1} \sim a^j b b^{i-1}$. Tedy slova $u = a^i b^i$ a $v = a^j b^i$ patří do stejné třídy rozkladu $\{a, b\}^*/_\sim$, přitom u náleží do jazyka L, zatímco v nikoliv. □

Poslední pojem, který budeme k formulaci Myhillovy-Nerodovy věty potřebovat, je obsažen v následující definici:

Definice 2.25. Nechť L je libovolný (ne nutně regulární) jazyk nad abecedou Σ. Na množině Σ^* definujeme relaci $\sim_L$ zvanou *prefixová ekvivalence pro* L takto:

$$u \sim_L v \overset{def}{\iff} \forall w \in \Sigma^* : (uw \in L \iff vw \in L) \ .$$

Tedy $\sim_L$ obsahuje právě ty dvojice (u, v), které mají tu vlastnost, že po připojení libovolného w, vzniklá slova uw, vw budou do jazyka L patřit buď obě, nebo ani jedno z nich.

Lemma 2.26. *Nechť L je libovolný jazyk nad Σ. Pak relace $\sim_L$ je pravá kongruence a L lze vyjádřit jako sjednocení některých (ne nutně konečně mnoha) tříd rozkladu $\Sigma^*/_{\sim_L}$.*

Důkaz. Zřejmě $\sim_L$ je ekvivalence. Dokážeme, že $\sim_L$ je pravá kongruence. Nechť $u \sim_L v$ a $a \in \Sigma$. Platí $ua \sim_L va$, neboť pro libovolné slovo $w \in \Sigma^*$ je $uaw \in L \iff vaw \in L$ (vyplývá to z toho, že aw je rovněž slovo nad Σ a $u \sim_L v$ – viz definice 2.25).

Zbývá dokázat, že L je sjednocením některých tříd rozkladu $\Sigma/_{\sim_L}$. K tomu stačí ukázat, že pro libovolná $u, v \in \Sigma^*$ platí $u \sim_L v \implies (u \in L \iff v \in L)$, tedy že slova v každé třídě patří do L buď všechna, nebo tam nepatří žádné z nich. Nechť tedy $u \sim_L v$. Podle definice 2.25 pak pro každé $w \in \Sigma^*$ platí $uw \in L \iff vw \in L$. Zvolíme-li za w prázdné slovo ε, obržíme $u\varepsilon = u \in L \iff v\varepsilon = v \in L$, což bylo dokázat. □

Každý jazyk nad abecedou Σ lze podle předchozího lemmatu vyjádřit jako sjednocení některých tříd rozkladu určeného jistou pravou kongruencí na Σ^* (těchto tříd však obecně nemusí být konečně mnoho). Pozorný čtenář patrně namítne, že za účelem konstatování tohoto faktu nebylo nutné zavádět relaci $\sim_L$, protože např. také identická relace *id* je pravá kongruence a každý jazyk lze vyjádřit jako sjednocení jistých tříd rozkladu $\Sigma^*/_{id}$. Relace $\sim_L$ však přece jen *je* něčím zvláštní:

Lemma 2.27. *Nechť L je jazyk nad abecedou Σ. Pro libovolnou pravou kongruenci $\sim$ na Σ^* takovou, že L je sjednocením některých tříd rozkladu $\Sigma^*/_\sim$ platí, že $\sim \subseteq \sim_L$, tj. $\sim_L$ je* největší *pravá kongruence s touto vlastností. Též říkáme, že každá $\sim$ je* zjeměním *$\sim_L$.*

Důkaz. Nechť $u \sim v$. Ukážeme, že pak také $u \sim_L v$, tj. pro libovolné slovo $w \in \Sigma^*$ platí $uw \in L \iff vw \in L$. K tomu si stačí uvědomit, že $uw \sim vw$ (viz definice 2.20); jelikož L je sjednocením některých tříd $\Sigma^*/_\sim$, platí $uw \in L \iff vw \in L$. □

Věta 2.28 (Myhillova-Nerodova). *Nechť L je jazyk nad Σ, pak tato tvrzení jsou ekvivalentní:*

1. *L je rozpoznatelný konečným automatem.*
2. *L je sjednocením některých tříd rozkladu určeného pravou kongruencí na Σ^* s konečným indexem.*
3. *Relace $\sim_L$ má konečný index.*

Důkaz. ($\mathbf{1} \Longrightarrow \mathbf{2}$) Viz věta 2.22 (ve směru 2.22–1$\Longrightarrow$ 2.22–2).

($\mathbf{2} \Longrightarrow \mathbf{3}$) Nechť $\sim$ je libovolná pravá kongruence na Σ^* s konečným indexem taková, že L je sjednocením některých tříd rozkladu $\Sigma^*/_\sim$. Protože $\sim$ je zjemněním $\sim_L$ (viz lemma 2.27), pak rozklad $\Sigma^*/_{\sim_L}$ má nejvýše tolik tříd jako $\Sigma^*/_\sim$, a tedy $\sim_L$ má též konečný index.

($\mathbf{3} \Longrightarrow \mathbf{1}$) Dle lemmatu 2.26 je L sjednocením některých tříd rozkladu $\Sigma^*/_{\sim_L}$ a $\sim_L$ je pravá kongruence. Jelikož $\sim_L$ má konečný index, lze opět použít větu 2.22 (tentokrát ve směru 2.22–2$\Longrightarrow$ 2.22–1), resp. druhou část důkazu Věty 2.22. □

2.1.4 Minimální konečný automat

Konečné automaty nacházejí velmi široké uplatnění v technické praxi (viz část 2.4). Z hlediska efektivity a nákladnosti implementace je důležité, aby počet stavů byl pokud možno co nejmenší. Přirozeným problémem je proto konstrukce tzv. *minimálního* automatu, tj. automatu *s nejmenším počtem stavů a totální přechodovou funkcí*, který rozpoznává daný regulární jazyk L. V této části ukážeme, že minimální automat lze sestrojit poměrně jednoduchým způsobem – stačí mít k dispozici *nějaký* konečný automat, který rozpoznává L. Minimální automat pak obdržíme ztotožněním některých jeho stavů.

Pozorný čtenář jistě postřehl, že tvrzení o existenci minimálního automatu a do jisté míry i návod k jeho konstrukci, jsou skryty v Myhillově-Nerodově větě (specielně viz důkaz implikace ($3 \Longrightarrow 1$) v 2.28, tj. konstrukce z druhé části důkazu věty 2.22 aplikované na $\sim_L$). Myhillovu-Nerodovu větu můžeme totiž reformulovat takto:

Věta 2.29 (Myhillova-Nerodova, 2. varianta). *Počet stavů libovolného minimálního automatu rozpoznávajícího jazyk L je roven indexu prefixové ekvivalence $\sim_L$. (Takový konečný automat existuje právě když index $\sim_L$ je konečný.)*

Důkaz. Víme, že každý konečný automat (a bez újmy na obecnosti bez nedosažitelných stavů) určuje jistou pravou kongruenci s konečným indexem a obráceně (viz věta 2.22). Je-li jazyk L regulární, pak relace $\sim_L$ je *největší* pravá kongruence s konečným indexem taková, že L je sjednocením některých tříd příslušného rozkladu (viz lemma 2.27). Konečný automat, který odpovídá relaci $\sim_L$, je tedy *minimální* automat rozpoznávající jazyk L a získáme jej tak, že aplikujeme postup uvedený v druhé části důkazu věty 2.22, tentokrát však nikoli pro $\sim$, ale pro $\sim_L$.

Jen pro zopakování připomeňme princip konstrukce: má-li $\sim_L$ konečný index k, konstruujeme FA $\mathcal{M}$ s k stavy. $\mathcal{M}$ si ve své konečné množině stavů uchovává informaci o tom, do které třídy ekvivalence dosud přečtená část vstupu patří. Při značení jako v důkazu 2.22 má tedy množinu stavů $\{[u] \mid u \in \Sigma^*\}$ o právě k prvcích (kde $[u] = \{u' \mid u' \sim_L u\}$). Stav $[u]$ je koncový, je-li $u \in L$, jinak není koncový. Přechodová fuknce je definována jako $\delta([u], a) = [ua]$. Důkaz korektnosti této konstrukce – viz 2.22. □

Příklad 2.30. *Myhillovu-Nerodovu větu lze, tak jako větu 2.22, použít k důkazu, že jazyk je či není akceptovatelný nějakým FA. Pro srovnání dokažme o témže jazyku jako v příkladu 2.15 a v příkladu 2.24, tj. $L = \{a^i b^i \mid i \geq 0\}$, že není regulární, a to pomocí 2.29 (tj. opět „jen" ilustrujeme důkazovou techniku implikovanou tvrzením této věty; čtenáři doporučujeme tyto techniky vzájemně porovnat).*

Žádné řetězy $\varepsilon, a, a^2, \ldots$ nejsou ekvivaletní vzhledem k $\sim_L$, protože předpoklad $a^i \sim_L a^j$ pro $i \neq j$ vede okamžitě ke sporu $a^i b^i \in L$, ale $a^j b^i \notin L$. Tedy $\sim_L$ má nekonečný index; jinak řečeno, L nemůže být rozpoznáván žádným konečným automatem.

Tvrzení o existenci minimálního konečného automatu můžeme tedy explicitněji zformulovat (jako bezprostřední důsledek Myhillovy-Nerodovy věty) takto:

Důsledek 2.31. *Minimální konečný automat akceptující jazyk L je určen jednoznačně až na isomorfismus (tj. přejmenování stavů).*

2.1.5 Minimalizace konečných automatů

Věnujme se nyní problému, jak k danému automatu *algoritmicky* nalézt ekvivaletní minimální automat. Zopakujme, že máme-li k dispozici nějaký konečný automat $\mathcal{M}$ rozpoznávající L, je příslušná pravá kongruence $\sim$ zjemněním relace $\sim_L$. Rozklad $\Sigma^*/\sim_L$ tedy vznikne z $\Sigma^*/\sim$ sjednocením některých tříd. Jelikož třídy $\Sigma^*/\sim_L$ odpovídají stavům minimálního automatu a třídy $\Sigma^*/\sim$ odpovídají stavům $\mathcal{M}$, můžeme také říci, že stavy minimálního automatu vzniknou ztotožněním některých stavů automatu $\mathcal{M}$. Zbývá zjistit, jak uvedené ztotožnění provést, a to samozřejmě beze změny akceptovaného jazyka.

Jistě nelze ztotožnit nějaký koncový stav p s nekoncovým stavem q. Pokud totiž $p = \hat{\delta}(q_0, x)$ a $q = \hat{\delta}(q_0, y)$, pak x musí být akceptován a y zamítnut, a to i po ztotožnění p a q. Není však způsob, jak zajistit, že „ztotožněný" stav má někdy akceptovat a někdy zamítat. Dále, pokud bychom ztotožnili nějaké p a q, pak bychom měli ztotožnit i jejich následníky

$\delta(p, a)$ a $\delta(q, a)$, abychom dodrželi funkcionalitu (tzv. determinismus) δ: pro daný stav a symbol je jednoznačně určen následník. Z těchto dvou úvah plyne, že nemůžeme ztotožnit p a q, pokud $\hat{\delta}(p, x) \in F$ a současně $\hat{\delta}(q, x) \notin F$ pro nějaké x. Ukazuje se, že tato podmínka je nutná i postačující pro rozhodování, kdy dva stavy ztotožnit, tj. pokud pro nějaké x $\hat{\delta}(p, x) \in F$ a současně $\hat{\delta}(q, x) \notin F$, pak stavy nemůžeme ztotožnit; pokud žádné takové x neexistuje, pak je ztotožnit můžeme. Tyto úvahy lze formalizovat takto:

Definice 2.32. Nechť $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ je FA bez nedosažitelných stavů a s totální přechodovou funkcí. Pro každý stav $q \in Q$ definujeme jazyk $L_{\mathcal{M}}(q) \subseteq \Sigma^*$, značený též jen $L(q)$, pokud $\mathcal{M}$ je zřejmý z kontextu, takto:

$$L(q) = \{x \in \Sigma^* \mid \hat{\delta}(q, x) \in F\} .$$

Stavy p, q nazveme *jazykově ekvivalentní*, psáno $p \equiv q$, pokud $L(p) = L(q)$, tj.:

$$p \equiv q \iff \forall x \in \Sigma^* : (\hat{\delta}(p, x) \in F \iff \hat{\delta}(q, x) \in F)$$

$L(q)$ je tedy jazyk přijímaný automatem $\mathcal{M}_q$, který vznikne z $\mathcal{M}$ tak, že za počáteční stav prohlásíme q. Zřejmě $\equiv$ je ekvivalence na Q. Intuitivně je jasné, že ztotožněním jazykově ekvivalentních stavů se přijímaný jazyk nezmění. K přesné formulaci tohoto faktu nejprve potřebujeme vědět, co se přesně myslí „ztotožněním stavů“. (Čtenář, kterému je alespoň intuitivně jasné, jak by zkonstruoval ekvivalentní automat s množinou stavů $Q/_{\equiv}$, pokud by byla dána $\equiv$, a že takto získaný automat $\mathcal{M}/_{\equiv}$ je minimální, může při prvním čtení přeskočit text až za důkaz věty 2.37, kde se věnujeme problému, jak spočítat $\equiv$.)

Lemma 2.33. *Nechť $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ je FA bez nedosažitelných stavů s totální přechodovou funkcí. Jestliže $p \equiv q$, pak pro každé $a \in \Sigma$ platí $\delta(p, a) \equiv \delta(q, a)$.*

Důkaz. Označme $r = \delta(p, a)$, $s = \delta(q, a)$. Potřebujeme dokázat, že $L(r) = L(s)$, tj. že pro libovolné slovo $w \in \Sigma^*$ platí $\hat{\delta}(r, w) \in F \iff \hat{\delta}(s, w) \in F$. K tomu si stačí uvědomit, že $\hat{\delta}(r, w) = \hat{\delta}(\delta(p, a), w) = \hat{\delta}(p, aw)$ a podobně $\hat{\delta}(s, w) = \hat{\delta}(\delta(q, a), w) = \hat{\delta}(q, aw)$. Zřejmě $\hat{\delta}(p, aw) \in F \iff \hat{\delta}(q, aw) \in F$, neboť $p \equiv q$. □

Definice 2.34. Nechť $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ je FA bez nedosažitelných stavů s totální přechodovou funkcí. *Reduktem* (též podílovým či faktor automatem) automatu $\mathcal{M}$ nazveme konečný automat $\mathcal{M}/_{\equiv} = (Q/_{\equiv}, \Sigma, \eta, [q_0], F/_{\equiv})$, tj. automat, kde

- Stavy jsou třídy rozkladu $Q/_{\equiv}$ (třídu obsahující stav q značíme $[q]$).
- Přechodová funkce η je definována pomocí reprezentantů. Je to nejmenší funkce splňující:

  $$\forall p, q \in Q, \forall a \in \Sigma : \ \delta(q, a) = p \implies \eta([q], a) = [p] .$$

  Aby tato definice byla korektní, nesmí záviset na volbě reprezentantů – pro každé dva stavy q, q' a každé $a \in \Sigma$ musí platit, že pokud $q \equiv q'$, pak také $\delta(q, a) \equiv \delta(q', a)$. To je však splněno podle lemmatu 2.33.
- Počáteční stav je třída rozkladu $Q/_{\equiv}$, obsahující stav q_0.
- Koncové stavy jsou právě ty třídy rozkladu $Q/_{\equiv}$, obsahující alespoň jeden koncový stav (jednoduchým důsledkem definice 2.32 je, že v každé třídě jsou koncové buď všechny stavy, nebo ani jeden z nich – tento fakt ospravedlňuje použité značení $F/_{\equiv}$).

Vztah mezi přechodovou funkcí δ automatu $\mathcal{M}$ a přechodovou funkcí η automatu $\mathcal{M}/_\equiv$ si zasluhuje bližší pozornosti:

Lemma 2.35. *Nechť $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ je konečný automat bez nedosažitelných stavů s totální přechodovou funkcí a $\mathcal{M}/_\equiv = (Q/_\equiv, \Sigma, \eta, [q_0], F/_\equiv)$ jeho redukt. Pro každé $u, v, w \in \Sigma^*$ platí:*

1. $\hat{\eta}([q_0], w) = [q] \iff \hat{\delta}(q_0, w) = p$, *kde* $p \equiv q$ *a*
2. $\hat{\eta}([q_0], u) = \hat{\eta}([q_0], v) \iff \hat{\delta}(q_0, u) \equiv \hat{\delta}(q_0, v)$

Důkaz. $(\mathbf{1})$: Indukcí vzhledem k délce slova w:

- $|\mathbf{w}| = \mathbf{0}$: Zřejmě $\hat{\delta}(q_0, \varepsilon) = q_0$ a platí $\hat{\eta}([q_0], \varepsilon) = [q] \iff q \equiv q_0$. Dokazovaná ekvivalence je tedy pravdivá.
- **Indukční krok:** Nechť $w = va$, kde $v \in \Sigma^*$, $a \in \Sigma$. Platí $\hat{\eta}([q_0], va) = [q]$ $\iff \hat{\eta}([q_0], v) = [r]$ a $\eta([r], a) = [q] \iff \hat{\delta}(q_0, v) = s$ kde $s \equiv r$ (indukční předpoklad) a $\delta(s, a) = p$, kde $p \equiv q$ (uvědomme si, že η nezávisí na volbě reprezentantů; jelikož $s \equiv r$ a $\delta(r, a) \equiv q$, musí také platit $\delta(s, a) \equiv q$) $\iff \hat{\delta}(q_0, va) = p$, kde $p \equiv q$.

$(\mathbf{2})$: Plyne přímo z tvrzení 1 tohoto lemmatu a definice množiny stavů automatu $\mathcal{M}/_\equiv$. □

Následující věta formálně vyjadřuje, že ztotožnění jazykově ekvivalentních stavů nezmění přijímaný jazyk:

Věta 2.36. *Nechť $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ je konečný automat bez nedosažitelných stavů s totální přechodovou funkcí. Pak $L(\mathcal{M}) = L(\mathcal{M}/_\equiv)$.*

Důkaz. Podle první části lemmatu 2.35 platí $\hat{\delta}(q_0, w) \in F \iff \hat{\eta}([q_0], w) \in F/_\equiv$, proto $L(\mathcal{M}) = L(\mathcal{M}/_\equiv)$. □

Nyní již lze dokázat hlavní výsledek této části:

Věta 2.37. *Nechť $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ je konečný automat bez nedosažitelných stavů s totální přechodovou funkcí, rozpoznávající jazyk L. Pak $\mathcal{M}/_\equiv$ je minimální automat rozpoznávající jazyk L.*

Důkaz. Dokážeme, že pravá kongruence $\sim$ určená automatem $\mathcal{M}/_\equiv$ ve smyslu věty 2.22 je přesně relace $\sim_L$. Pro libovolná $u, v \in \Sigma^*$ platí:

$$
\begin{aligned}
u \sim v &\iff \hat{\eta}([q_0], u) = \hat{\eta}([q_0], v) \quad \text{dle algoritmu z Nerodovy věty}\\
&\iff \hat{\delta}(q_0, u) \equiv \hat{\delta}(q_0, v) \quad \text{dle lemmatu 2.35}\\
&\iff (\forall w \in \Sigma^* : \hat{\delta}(\hat{\delta}(q_0, u), w) \in F \iff \hat{\delta}(\hat{\delta}(q_0, v), w) \in F) \quad \text{dle definice } \equiv\\
&\iff (\forall w \in \Sigma^* : \hat{\delta}(q_0, uw) \in F \iff \hat{\delta}(q_0, vw) \in F)\\
&\iff (\forall w \in \Sigma^* : uw \in L \iff vw \in L)\\
&\iff u \sim_L v \quad \text{dle definice } \sim_L .
\end{aligned}
$$

Dokázali jsme tedy, že pro všechna $u, v \in \Sigma^*$ platí $u \sim v \iff u \sim_L v$, takže $\sim\, = \,\sim_L$, což bylo dokázat. □

Předchozí věta sice říká, jak pro daný konečný automat $\mathcal{M}$ vypadá ekvivalentní minimální automat, ale nepodává algoritmus pro jeho konstrukci – není totiž na první pohled jasné, jakým způsobem lze zkonstruovat relaci $\equiv$. Tímto problémem se budeme nyní zabývat.

Definice 2.32 říká, že $p \equiv q$ pokud pro každé slovo w platí, že $\hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F$. Relaci $\equiv$ lze tedy velmi přirozeně aproximovat tak, že položíme omezení na délku slova w:

Definice 2.38. Nechť $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ je konečný automat bez nedosažitelných stavů, jehož přechodová funkce je totální. Pro každé $i \in \mathbb{N}_0$ definujeme binární relaci $\equiv_i$ na Q předpisem (srv. s definicí relace $\equiv$ v 2.32):

$$p \equiv_i q \overset{def}{\iff} \forall w \in \Sigma^*. |w| \leq i : (\hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F)$$

Pokud $p \equiv_i q$, znamená to, že stavy p a q nelze „rozlišit" ve smyslu definice 2.32 žádným slovem délky nejvýše i. Tedy $p \equiv q$ právě když $p \equiv_i q$ pro každé $i \in \mathbb{N}_0$. Množinovou symbolikou to lze vyjádřit takto:

$$\equiv \; = \bigcap_{i \geq 0} \equiv_i \tag{2.4}$$

Zřejmě každá z relací $\equiv_i$ je ekvivalence na Q a navíc $\equiv_{i+1}$ je zjemněním $\equiv_i$ pro každé $i \in \mathbb{N}_0$. Následující lemma obsahuje návod, jak relace $\equiv_i$ počítat. Jedná se de facto o induktivní předpis vzhledem k délce rozlišujících slov. Musíme též ukázat korektnost tohoto předpisu pro výpočet $\equiv_i$ vůči Definici 2.38.

Lemma 2.39. *Pro relace $\equiv_i$ platí:*

1. $p \equiv_0 q \iff (p \in F \iff q \in F)$
2. $p \equiv_{i+1} q \iff p \equiv_i q \;\wedge\; \forall a \in \Sigma : \delta(p, a) \equiv_i \delta(q, a)$

Důkaz. Tvrzení dokážeme indukcí vzhledem k i. Báze (případ $i = 0$) zřejmě platí, neboť slovem ε lze ve smyslu definice 2.38 rozlišit pouze koncové a nekoncové stavy.

Předpokládejme, že tvrzení 2 platí pro nějaké i a ukažme jeho platnost pro $i + 1$.
Zřejmě platí (*): $p \equiv_{i+1} q$ (tj. nejsou rozlišitelné žádným slovem délky nejvýše $i + 1$)
$\iff$ (i) $p \equiv_i q$ (tj. nejsou rozlišitelné žádným slovem délky nejvýše i) a
(ii) $\forall w. |w| = i + 1$ platí $\hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F$
(tj. a nejsou rozlišitelné ani žádným slovem délky $i + 1$).
Podmínku (ii) lze ekvivalentně zapsat takto:
$\forall w \in \Sigma^{i+1}. (\hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F)$, což platí právě když
$\forall a \in \Sigma. \forall v \in \Sigma^i. (\hat{\delta}(p, av) \in F \iff \hat{\delta}(q, av) \in F) \iff$
$\forall a \in \Sigma. \forall v \in \Sigma^i. (\hat{\delta}(\delta(p, a), v) \in F \iff \hat{\delta}(\delta(q, a), v) \in F) \iff$
$\forall a \in \Sigma. (\delta(p, a) \equiv_i \delta(q, a))$.
Z výše uvedeného tvrzení (*) a právě dokázaného ekvivalentního zápisu podmínky (ii) tedy dostáváme požadované: $p \equiv_{i+1} q \iff p \equiv_i q \;\wedge\; \forall a \in \Sigma. (\delta(p, a) \equiv_i \delta(q, a))$. $\square$

Označme n počet stavů automatu $\mathcal{M}$. Připomeňme že, každá $\equiv_i$ je relací evivalence a navíc $\equiv_{i+1}$ je zjemněním $\equiv_i$ pro každé $i \in \mathbb{N}_0$, tj. $\equiv_{i+1}$ má alespoň tolik tříd rozkladu jako $\equiv_i$. Jelikož libovolná ekvivalence na n-prvkové množině může mít nejvýše n tříd a $\equiv_0$ má aspoň jednu třídu, pak nutně (podle Dirichletova principu) platí, že musí existovat nějaké $k \leq n - 1$ takové, že $\equiv_k$ a $\equiv_{k+1}$ mají stejný počet tříd, což (opět díky faktu o zjemnění) dává $\equiv_k = \equiv_{k+1}$. To ovšem znamená, že dokonce $\equiv_k = \equiv_{k+j}$ pro libovolné $j \in \mathbb{N}_0$, neboť relace $\equiv_{i+1}$ závisí pouze na relaci $\equiv_i$. Vztah (2.4) je tedy možné přepsat jako

$$\equiv\; = \bigcap_{i=0}^{k} \equiv_i \;=\; \equiv_k \tag{2.5}$$

Tím je formálně dokázána správnost i konečnost algoritmu 2.2 pro minimalizaci konečného automatu. Po inicializaci ($i = 0$) iterativně počítáme $\equiv_i$, $i = 1, 2, \ldots$ a skončíme pro $i = k$ takové, že platí $\equiv_k \;=\; \equiv_{k-1}$.

Algoritmus 2.2 Minimalizace konečného automatu.

Vstup: Konečný automat $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ bez nedosažitelných stavů s totální přechodovou funkcí.

Výstup: Redukt $\mathcal{M}/_{\equiv}$.

$i := 0$;
$\equiv_0 := \{(p, q) \mid p \in F \Longleftrightarrow q \in F\}$;
repeat
  $\equiv_{i+1} := \{(p, q) \mid p \equiv_i q \;\wedge\; \forall a \in \Sigma : \delta(p, a) \equiv_i \delta(q, a)\}$;
  $i := i + 1$;
until $\equiv_i \;=\; \equiv_{i-1}$
$\equiv \;:=\; \equiv_i$;
$\mathcal{M}/_{\equiv} := (Q/_{\equiv}, \Sigma, \eta, [q_0], F/_{\equiv})$;

Příklad 2.40. *Mějme konečný automat $\mathcal{M}$ daný níže uvedenou tabulkou. Při konstrukci minimálního automatu je nejprve třeba odstranit nedosažitelné stavy a zúplnit přechodovou funkci. Obdržíme tak automat $\mathcal{M}'$ (stav 7 byl nedosažitelný, za účelem zúplnění přechodové funkce byl dále přidán nový stav N):*

	$\mathcal{M}$	a	b
$\rightarrow$	1	2	–
	2	3	4
$\leftarrow$	3	6	5
	4	3	2
$\leftarrow$	5	6	3
$\leftarrow$	6	2	–
	7	6	1

	$\mathcal{M}'$	a	b
$\rightarrow$	1	2	N
	2	3	4
$\leftarrow$	3	6	5
	4	3	2
$\leftarrow$	5	6	3
$\leftarrow$	6	2	N
	N	N	N

Nyní již lze přistoupit ke konstrukci relací $\equiv_i$. Technicky to lze provést například tak, že v tabulce přechodové funkce sdružíme řádky odpovídající stavům, které jsou v relaci $\equiv_i$ a jednotlivé skupiny (třídy rozkladu $Q/_{\equiv_i}$) označíme římskými číslicemi. Aby bylo možné snadno určit následující relaci $\equiv_{i+1}$, doplníme do každého políčka (q, x) číslo třídy, do níž patří stav $\delta(q, x)$ (dvojice (q, x) označuje políčko na řádku q ve sloupci x). Třídy rozkladu určeného $\equiv_{i+1}$ pak lze získat tak, že existující skupiny dále rozdělíme – v rámci každé skupiny sdružíme stavy, které mají řádky vyplněné stejným způsobem. Pokud dojde k tomu, že *v každé skupině mají všechny stavy řádky vyplněné stejně, není již důvod něco rozdělovat; nalezli jsme relaci* $\equiv$.

	$\equiv_0$	a	b
I	1	*I*	*I*
	2	*II*	*I*
	4	*II*	*I*
	N	*I*	*I*
II	3	*II*	*II*
	5	*II*	*II*
	6	*I*	*I*

	$\equiv_1$	a	b
I	1	*II*	*I*
	N	*I*	*I*
II	2	*III*	*II*
	4	*III*	*II*
III	3	*IV*	*III*
	5	*IV*	*III*
IV	6	*II*	*I*

	$\equiv_2$	a	b
I	1	*III*	*II*
II	N	*II*	*II*
III	2	*IV*	*III*
	4	*IV*	*III*
IV	3	*V*	*IV*
	5	*V*	*IV*
V	6	*III*	*II*

Relace $\equiv$ *je tedy v tomto případě rovna relaci* $\equiv_2$. *Minimální automat pro jazyk* $L(\mathcal{M})$ *vypadá takto:*

	$\mathcal{M}/\equiv$	a	b
$\rightarrow$	*I*	*III*	*II*
	II	*II*	*II*
	III	*IV*	*III*
$\leftarrow$	*IV*	*V*	*IV*
$\leftarrow$	*V*	*III*	*II*

2.2 Konservativní rozšíření modelu konečných automatů

V této části si ukážeme některé způsoby, jak lze základní model konečného automatu dále rozšířit či zobecnit bez toho, aby se změnila výpočetní síla – tj. ke každému rozšířenému modelu (akceptujícímu nějaký jazyk) bude existovat model základní s ním jazykově ekvivaletní (akceptující týž jazyk). Takovéto rozšíření se nazývá konservativní.

2.2.1 Nedeterministické konečné automaty

Nedeterministický konečný automat je zařízení, které je velmi podobné konečnému automatu z definice 2.1. Jediný rozdíl je v tom, že nedeterministický automat nemusí mít pro daný stav a vstupní symbol určen následující stav jednoznačně (na přechodových grafech si to lze představit tak, že z jednoho uzlu může vycházet více hran se stejným návěštím). Není tedy předem jasné, do jakého stavu se automat dostane po zpracování daného slova w, neboť automat si během výpočtu může „vybírat" jeden z možných následujících stavů. Slovo w bude akceptováno, pokud *alespoň jeden* z možných výpočtů nad slovem w skončí v koncovém stavu.

Poznámka 2.41. *V dalším textu budeme označovat konečné automaty z definice 2.1 přívlastkem* deterministické, *zkráceně DFA, abychom předešli nedorozumění.*

Definice 2.42. *Nedeterministický konečný automat*, zkráceně NFA, je $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$, kde význam všech složek je stejný jako v definici 2.1 s výjimkou přechodové funkce δ. Ta je definována jako (totální) zobrazení $\delta : Q \times \Sigma \to 2^Q$.

Na deterministické automaty tedy můžeme pohlížet jako na speciální případ nedeterministických automatů, kdy pro každé $q \in Q$ a $a \in \Sigma$ je množina $\delta(q, a)$ nejvýše jednoprvková.

Podobně jako v případě deterministických automatů zavedeme *rozšířenou* přechodovou funkci $\hat{\delta} : Q \times \Sigma^* \to 2^Q$:

- $\hat{\delta}(q, \varepsilon) = \{q\}$
- $\hat{\delta}(q, wa) = \bigcup_{p \in \hat{\delta}(q,w)} \delta(p, a)$

Jazyk přijímaný nedeterministickým konečným automatem $\mathcal{M}$ je definován takto:

$$L(\mathcal{M}) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$

Nedeterministické konečné automaty $\mathcal{M}$ a $\mathcal{M}'$ jsou *ekvivalentní*, pokud $L(\mathcal{M}) = L(\mathcal{M}')$.

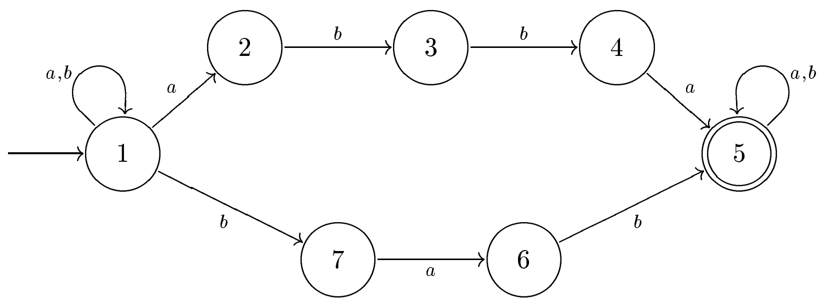
Stejně jako v případě deterministických konečných automatů je také možné definovat jazyk $L(\mathcal{M})$ pomocí pojmů konfigurace a krok výpočtu; jediná změna je v definici relace kroku výpočtu:

$$(q, aw) \ \vdash \ (p, w) \ \stackrel{def}{\Longleftrightarrow} \ p \in \delta(q, a)$$

Nedeterminismus je velmi silný popisný aparát, který často umožňuje zachytit strukturu jazyka elegantním a přirozeným způsobem. Uvažme např. jazyk

$$L = \{w \in \{a, b\}^* \mid w \text{ obsahuje podslovo } abba \text{ nebo } bab\}$$

Navrhnout deterministický automat, který rozpoznává L, není zcela triviální. Naopak nedeterministický automat lze zkonstruovat snadno:



Struktura automatu velmi přímočaře odráží fakt, že L je složen ze slov, která začínají nějakým řetězcem symbolů a, b, pak následuje jedno z podslov $abba, bab$ a na konci je zase nějaký řetězec složený z a, b.

Nedeterminismus lze dobře využít jako popisný prostředek, nemá však vliv na výpočetní sílu konečných automatů. Ke každému nedeterministickému konečnému automatu totiž ve skutečnosti existuje ekvivalentní deterministický automat, který lze dokonce algoritmicky zkonstruovat. Důkaz tohoto faktu se opírá o zajímavou techniku, které se říká *podmnožinová konstrukce*:

Věta 2.43. *Pro každý NFA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ existuje ekvivalentní DFA.*

Důkaz. Nechť $\mathcal{M}' = (Q', \Sigma, \delta', \{q_0\}, F')$ je deterministický konečný automat, kde:

- $Q' = 2^Q$, tj. stavy automatu $\mathcal{M}'$ jsou všechny podmnožiny Q.
- Přechodová funkce δ' je definována předpisem $\delta'(P, a) = \bigcup_{q \in P} \delta(q, a)$
- Množina koncových stavů F' je tvořena právě těmi podmnožinami Q, které obsahují alespoň jeden prvek množiny F.

Zřejmě $\mathcal{M}'$ je deterministický konečný automat (dokonce s totální přechodovou funkcí). Nejprve ukažme, že pro každé $w \in \Sigma^*$ platí $\hat{\delta}(q_0, w) = \hat{\delta}'(\{q_0\}, w)$. Indukcí k délce w:

- $|\mathbf{w}| = \mathbf{0}$: Platí $\hat{\delta}(q_0, \varepsilon) = \{q_0\} = \hat{\delta}'(\{q_0\}, \varepsilon)$.
- **Indukční krok:** Nechť $w = va$, kde $v \in \Sigma^*, a \in \Sigma$. Pak platí: $\hat{\delta}(q_0, va) = \bigcup_{p \in \hat{\delta}(q_0, v)} \delta(p, a) = \delta'(\hat{\delta}(q_0, v), a)$ (dle definice δ') $= \delta'(\hat{\delta}'(\{q_0\}, v), a)$ (dle indukčního předpokladu) $= \hat{\delta}'(\{q_0\}, va)$.

Nyní se již snadno vidí, že $L(\mathcal{M}) = L(\mathcal{M}')$, neboť $w \in L(\mathcal{M}) \iff \hat{\delta}(q_0, w) \cap F \neq \emptyset \iff \hat{\delta}'(\{q_0\}, w) \cap F \neq \emptyset \iff \hat{\delta}'(\{q_0\}, w) \in F' \iff w \in L(\mathcal{M}')$. □

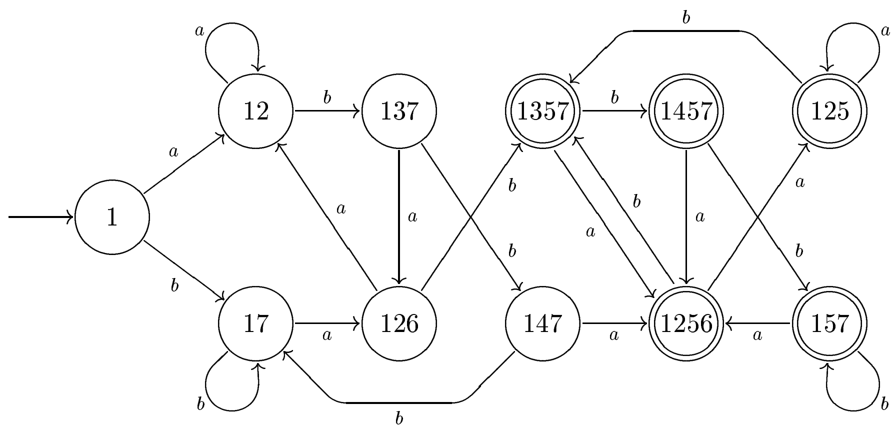
Algoritmus 2.3 Transformace NFA na ekvivalentní DFA.

Vstup: Nedeterministický konečný automat $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$.
Výstup: Ekvivalentní deterministický konečný automat $\mathcal{M} = (Q', \Sigma, \delta', \{q_0\}, F')$ bez nedosažitelných stavů s totální přechodovou funkcí.

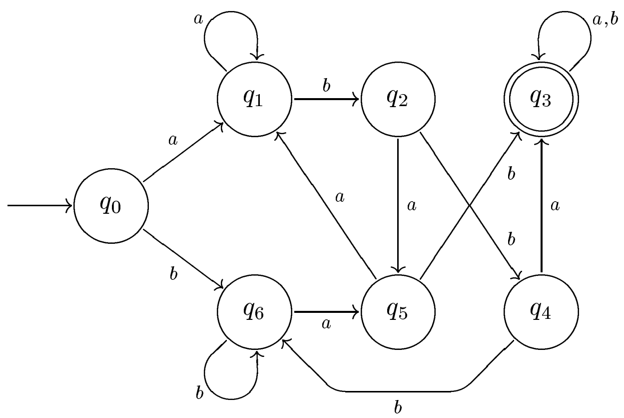
$Q' := \{\{q_0\}\}$; $\delta' := \emptyset$; $F' := \emptyset$; *Done* $:= \emptyset$;
while $(Q' - \mathit{Done}) \neq \emptyset$ **do**
  $M :=$ libovolný prvek množiny $Q' - \mathit{Done}$;
  if $M \cap F \neq \emptyset$ **then**
    $F' := F' \cup \{M\}$;
  end if
  for all $a \in \Sigma$ **do**
    $N := \bigcup_{p \in M} \delta(p, a)$;
    $Q' := Q' \cup \{N\}$;
    $\delta' := \delta' \cup \{((M, a), N)\}$;
  end for
  $\mathit{Done} := \mathit{Done} \cup \{M\}$;
end while
$\mathcal{M}' := (Q', \Sigma, \delta', \{q_0\}, F')$;

Uvedený důkaz je konstruktivní (podává návod na sestrojení automatu $\mathcal{M}'$). Nevýhodou prezentovaného algoritmu však je, že automat $\mathcal{M}'$ může obsahovat mnoho nedosažitelných stavů. Tento nedostatek lze odstranit jednoduše – iterativní konstrukcí množiny dosažitelných stavů a přechodové funkce. Obdržíme tak algoritmus 2.3.

Aplikujeme-li algoritmus 2.3 na dříve uvedený nedeterministický automat, rozpoznávající jazyk $L = \{w \in \{a, b\}^* \mid w$ obsahuje podslovo $abba$ nebo $bab\}$, dostaneme tento výstup:



Tento příklad také demonstruje, že výstupem algoritmu 2.3 nemusí být minimální automat; ten totiž pro jazyk L vypadá takto:



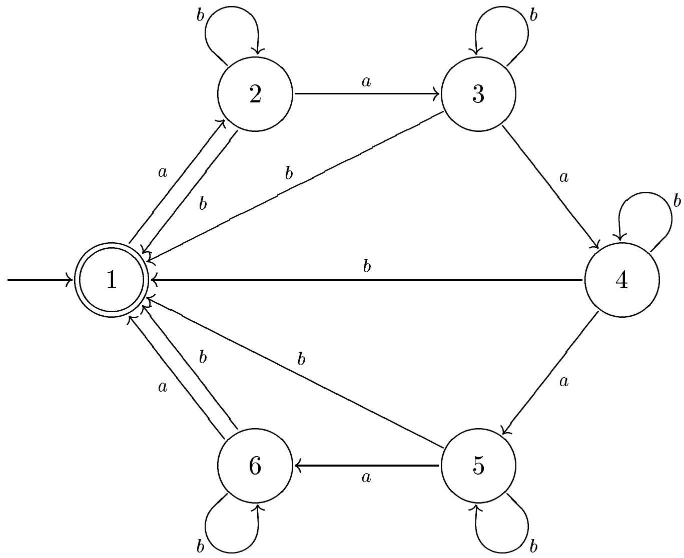
Čtenář se nyní může pokusit zjistit, jaká informace o přečtené části vstupního slova je spojena s jednotlivými stavy.

Podmnožinová konstrukce je na první pohled značně neefektivní – nárůst počtu stavů při přechodu od nedeterministického konečného automatu k deterministickému je exponenciální. To je z technického hlediska nepříjemné. Nabízí se proto otázka, zda použitím „pomocných“ technik (odstranění nedosažitelných stavů, minimalizace) je obecně možné dosáhnout lepšího výsledku. Následující věta říká, že nikoliv.

Věta 2.44. *Pro každé $n \in \mathbb{N}$ existuje NFA o n stavech takový, že ekvivalentní DFA má i po minimalizaci 2^n stavů.*

Důkaz. Buď $n \in \mathbb{N}$ libovolné přirozené číslo. Zkonstruujeme nedeterministický konečný automat o n stavech, který má požadovanou vlastnost. Nechť $\mathcal{M} = (Q, \Sigma, \delta, 1, \{1\})$, kde

- $Q = \{1, \ldots, n\}$,
- $\Sigma = \{a, b\}$ a
- δ je nejmenší funkce splňující tyto podmínky:



Obrázek 2.2: Automat $\mathcal{M}$ z důkazu věty 2.44 pro $n = 6$.

- $\delta(i, a) = \{i + 1\}$ pro $1 \leq i \leq n - 1$,
- $\delta(n, a) = \{1\}$,
- $\delta(i, b) = \{1, i\}$ pro $2 \leq i \leq n$.

Na obrázku 2.2 je znázorněn automat $\mathcal{M}$ pro $n = 6$. Dokážeme, že deterministický konečný automat $\mathcal{M}'$, který je výstupem algoritmu 2.3, má právě 2^n stavů a je minimální.

To, že $\mathcal{M}'$ má právě 2^n stavů plyne z toho, že libovolná podmnožina Q je dosažitelným stavem automatu $\mathcal{M}'$. Totiž prázdná množina je zřejmě dosažitelný stav, neboť $\hat{\delta}'(1, b) = \emptyset$. Dále nechť $\{k_1, \ldots, k_l\} \subseteq Q$ je neprázdná podmnožina. Předpokládejme, že $k_i < k_j$ pro $i < j$ a označme $d_i = k_{i+1} - k_i$ pro $1 \leq i < l$. Pak stav $\{k_1, \ldots, k_l\}$ je v automatu $\mathcal{M}'$ dosažitelný pod slovem $a^{d_{l-1}} b a^{d_{l-2}} b a^{d_{l-3}} \ldots a^{d_1} b a^{k_1 - 1}$ (tedy např. stav $\{2, 4, 5\}$ je v automatu na obrázku 2.2 dosažitelný pod slovem $abaaba$). Tento fakt lze lehce ověřit indukcí vzhledem k l.

Zbývá dokázat, že $\mathcal{M}'$ je minimální. K tomu stačí ověřit, že pro každé dva různé stavy U, V automatu $\mathcal{M}'$ platí $U \not\equiv V$ (viz definice 2.32). Jelikož $U \neq V$, existuje $k \in \{1, \ldots, n\}$, které patří do U ale nepatří do V, nebo obráceně. Slovem, kterým lze stavy U a V rozlišit ve smyslu definice 2.32, je a^{n-k+1}. $\square$

Tedy i pro poměrně „malý" nedeterministický konečný automat může být ekvivalentní deterministický automat „velmi velký" i po minimalizaci. V praxi je velikost stavového prostoru samozřejmě omezená použitou technologií. Znamená to tedy, že nedeterministické konečné automaty obecně nelze efektivně implementovat? Naštěstí tomu tak není; technický „trik", kterým lze exponenciální nárůst počtu stavů obejít, je skryt v samotné podmnožinové konstrukci: Nechť $Q = \{q_1, \ldots, q_n\}$ je množina stavů nedeterministického konečného automatu $\mathcal{M}$. Libovolnou podmnožinu $X \subseteq Q$ pak můžeme jednoznačně zakódovat bitovým řetězcem $\mathcal{B}$ délky n takto:

$$i\text{-tý prvek } \mathcal{B} = \begin{cases} 1 & \text{pokud } q_i \in X, \\ 0 & \text{jinak.} \end{cases}$$

Libovolný stav deterministického automatu $\mathcal{M}'$, který je výstupem podmnožinové konstrukce aplikované na $\mathcal{M}$, můžeme tedy reprezentovat n-bitovým řetězcem. Technická implementace automatu $\mathcal{M}'$ pak spočívá v realizaci dvou funkcí:

`function NextState(state: StateCode, symbol: Char): StateCode`

Tato funkce má dva parametry: `state` je stav automatu $\mathcal{M}'$, zakódovaný jako n-bitový řetězec (bitové řetězce jsou implementovány pomocí datového typu `StateCode`). Druhý parametr `symbol` je vstupní symbol. Funkce vrací kód stavu $\delta'(q, \texttt{symbol})$, kde q je stav s kódem `state`.

`function IsFinal(state: StateCode): Boolean`

Funkce má jeden parametr, kterým je stav automatu $\mathcal{M}'$, zakódovaný jako n-bitový řetězec. Vrací `True` pokud je stav s kódem `state` koncový, jinak `False`.

Jedinou datovou strukturou, která je potřebná pro implementaci těchto funkcí, je tabulka přechodové funkce δ původního nedeterministického automatu $\mathcal{M}$ s vyznačenými koncovými stavy.

Funkce `NextState` na základě svých parametrů a tabulky pro δ snadno určí kód výsledného stavu (i-tý bit výsledku bude nastaven na 1, právě když $q_i \in \delta(q_j, \texttt{symbol})$ pro nějaké j takové, že j-tý bit prvního parametru je nastaven na 1).

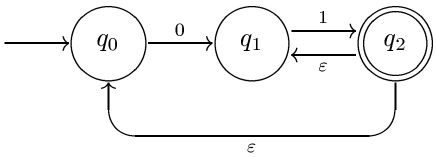
Podobně funkce `IsFinal` vrátí hodnotu `True`, pokud existuje i takové, že $q_i \in F$ a i-tý bit parametru je 1. Není tedy zapotřebí implementovat tabulku přechodové funkce δ', ani není nutné *explicitně* reprezentovat stavy automatu $\mathcal{M}'$.

Na nedeterministický automat $\mathcal{M}$ proto můžeme pohlížet jako na *symbolickou* reprezentaci automatu $\mathcal{M}'$, která dokáže popsat stavy i přechodovou funkci $\mathcal{M}'$ podstatně hutnější (úspornější) syntaxí.

2.2.2 Automaty s ε-kroky

Model nedeterministického konečného automatu je možné dále rozšířit o tzv. ε-kroky. Automat pak může svůj stav za určitých okolností změnit samovolně, tj. bez přečtení vstupního symbolu. Tato schopnost je formálně popsána pomocí ε-přechodů, které si lze na přechodových grafech představit jako hrany, jejichž návěštím je prázdné slovo. Díky těmto přechodům v definici přechodové funkce může automat během výpočtu na slově w provést jeden či více ε-kroků, tj. změnit svůj stav bez toho, aby ze vstupu cokoliv přečetl.

Příklad 2.45. *Následující automat s ε-kroky rozpoznává právě ta slova $w \in \{0,1\}^*$, která jsou tvaru $01^{k_1}01^{k_2}\dots01^{k_n}$, kde $n \geq 1$ a $k_i \geq 1$ pro každé $1 \leq i \leq n$:*



Definice 2.46. *Nedeterministický konečný automat s ε-kroky*, zkráceně εNFA, je pětice $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$, kde význam všech složek je stejný jako v definici 2.42 s výjimkou přechodové funkce δ. Ta je definována jako totální zobrazení $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$.

Rozšířenou přechodovou funkci $\hat{\delta}$ ovšem musíme definovat odlišným způsobem; nejprve zavedeme funkci $D_\varepsilon : Q \to 2^Q$, která pro daný stav p vrací množinu stavů, kterých může $\mathcal{M}$ dosáhnout z p bez toho, aby četl vstup. Pro dané $p \in Q$ je $D_\varepsilon(p)$ definována (induktivně) jako nejmenší množina $X \subseteq Q$ taková, že platí:

- $p \in X$,
- je-li $q \in X$ a $r \in \delta(q, \varepsilon)$, pak také $r \in X$.

Pro automat z příkladu 2.45 platí $D_\varepsilon(q_0) = \{q_0\}, D_\varepsilon(q_1) = \{q_1\}, D_\varepsilon(q_2) = \{q_0, q_1, q_2\}$. Funkci D_ε je rozšíříme na množiny stavů, tj. na $D_\varepsilon : 2^Q \to 2^Q$ takto: je-li $Y \subseteq Q$, položíme

$$D_\varepsilon(Y) = \bigcup_{q \in Y} D_\varepsilon(q)$$

Nyní již můžeme definovat rozšířenou přechodovou funkci $\hat{\delta} : Q \times \Sigma^* \to 2^Q$ takto:

- $\hat{\delta}(q, \varepsilon) = D_\varepsilon(q)$,
- $\hat{\delta}(q, wa) = \bigcup_{p \in \hat{\delta}(q,w)} D_\varepsilon(\delta(p, a))$.

Pro automat z příkladu 2.45 mimo jiné platí $\hat{\delta}(q_2, 1) = \{q_0, q_1, q_2\}, \hat{\delta}(q_1, 1) = \{q_0, q_1, q_2\}$ a $\hat{\delta}(q_2, 0) = \{q_1\}$.

Jazyk přijímaný automatem $\mathcal{M}$ s ε-kroky je definován stejně jako v případě nedeterministických automatů, tj.

$$L(\mathcal{M}) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$

Není obtížné dokázat, že ke každému automatu $\mathcal{M}$ s ε-kroky existuje ekvivalentní nedeterministický automat $\mathcal{M}'$; v principu je třeba v přechodovém grafu automatu $\mathcal{M}$ přidat hranu $p_1 \xrightarrow{a} q_n$ pro každou cestu tvaru

$$p_1 \xrightarrow{\varepsilon} \dots \xrightarrow{\varepsilon} p_m \xrightarrow{a} q_1 \xrightarrow{\varepsilon} \dots \xrightarrow{\varepsilon} q_n$$

kde $m, n \geq 1$, $a \in \Sigma$. Pak lze ε-hrany z přechodového grafu bez problémů odstranit. Nejprve dokážeme jedno pomocné tvrzení:

Lemma 2.47. *Nechť $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ je automat s ε-kroky. V přechodovém grafu automatu $\mathcal{M}$ existuje cesta $p_1 \xrightarrow{\varepsilon} \dots \xrightarrow{\varepsilon} p_m \xrightarrow{a} q_1 \xrightarrow{\varepsilon} \dots \xrightarrow{\varepsilon} q_n$, kde $m, n \geq 1$, $a \in \Sigma$, právě když $q_n \in \hat{\delta}(p_1, a)$.*

Důkaz. Nejprve si uvědomme, že podle definice $\hat{\delta}$ platí

$$\hat{\delta}(p_1, a) = \bigcup_{r \in D_\varepsilon(p_1)} D_\varepsilon(\delta(r, a)) \tag{2.6}$$

($\Longrightarrow$) Jelikož $p_1 \xrightarrow{\varepsilon} \dots \xrightarrow{\varepsilon} p_m$, platí $p_m \in D_\varepsilon(p_1)$. Dále $p_m \xrightarrow{a} q_1$, tedy $q_1 \in \delta(p_m, a)$. Konečně $q_1 \xrightarrow{\varepsilon} \dots \xrightarrow{\varepsilon} q_n$, proto $q_n \in D_\varepsilon(q_1)$. Celkem $q_n \in \hat{\delta}(p_1, a)$ podle vztahu (2.6).

($\Longleftarrow$) Nechť $q_n \in \hat{\delta}(p_1, a)$. Podle vztahu (2.6) pak musí existovat stav $r \in D_\varepsilon(p_1)$ takový, že $q_n \in D_\varepsilon(\delta(r,a))$. Jelikož $r \in D_\varepsilon(p_1)$, existuje v přechodovém grafu cesta $p_1 \xrightarrow{\varepsilon} \ldots \xrightarrow{\varepsilon} r$. Protože $q_n \in D_\varepsilon(\delta(r,a))$, musí existovat stav $s \in \delta(r,a)$ takový, že $q_n \in D_\varepsilon(s)$. V přechodovém grafu proto existuje hrana $r \xrightarrow{a} s$ a cesta $s \xrightarrow{\varepsilon} \ldots \xrightarrow{\varepsilon} q_n$. Celkem $p_1 \xrightarrow{\varepsilon} \ldots \xrightarrow{\varepsilon} r \xrightarrow{a} s \xrightarrow{\varepsilon} \ldots \xrightarrow{\varepsilon} q_n$, což bylo dokázat. □

Věta 2.48. *Ke každému NFA s ε-kroky existuje ekvivalentní NFA (bez ε-kroků).*

Důkaz. Nechť $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ je libovolný NFA s ε-kroky a sestrojme ekvivaletní NFA $\mathcal{M}' = (Q, \Sigma, \gamma, q_0, G)$ bez ε-kroků. Položme $\gamma(q,a) = \hat{\delta}(q,a)$ pro každé $q \in Q, a \in \Sigma$ a množinu G definujme takto:

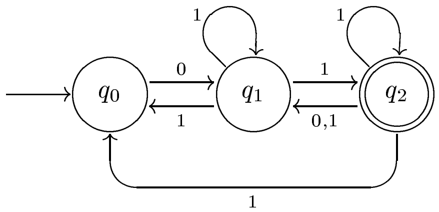
$$G = \begin{cases} F & \text{pokud } D_\varepsilon(q_0) \cap F = \emptyset, \\ F \cup \{q_0\} & \text{jinak.} \end{cases}$$

Nejprve dokážeme, že pro libovolné $p \in Q$, $w \in \Sigma^+$ platí $\hat{\delta}(p,w) = \hat{\gamma}(p,w)$, tj. $q \in \hat{\delta}(p,w) \iff q \in \hat{\gamma}(p,w)$. V této souvislosti je třeba si uvědomit, že funkce $\hat{\delta}, \hat{\gamma}$ jsou definovány na základě funkcí δ, γ odlišným způsobem – $\hat{\gamma}$ je rozšířenou přechodovou funkcí nedeterministického konečného automatu $\mathcal{M}'$, je tedy určena předpisem uvedeným za definicí 2.42, zatímco $\hat{\delta}$ je určena předpisem za definicí 2.46 (pro $w = \varepsilon$ tedy uvedená ekvivalence platit nemusí). Důkaz provedeme indukcí k délce slova w.

- $|\mathbf{w}| = \mathbf{1}$: Pak $w = a$ pro nějaké $a \in \Sigma$ a $\hat{\delta}(p,a) = \gamma(p,a) = \hat{\gamma}(p,a)$ podle definic γ a $\hat{\gamma}$.
- **Indukční krok:** Nechť $w = va$, kde $v \in \Sigma^+$ a $a \in \Sigma$. Platí $q \in \hat{\delta}(p,w) \iff$ existují stavy r, s takové, že $r \in \hat{\delta}(p,v)$, $s \in \delta(r,a)$ a $q \in D_\varepsilon(s)$ (existuje tedy cesta $r \xrightarrow{a} s \xrightarrow{\varepsilon} \ldots \xrightarrow{\varepsilon} q) \iff r \in \hat{\gamma}(p,v)$ (s využitím indukčního předpokladu), $q \in \hat{\delta}(r,a)$ (lemma 2.47) $\iff r \in \hat{\gamma}(p,v)$ a $q \in \gamma(r,a)$ (podle definice γ) $\iff q \in \hat{\gamma}(p,va)$.

Pro libovolné $w \in \Sigma^+$ tedy platí: $\hat{\delta}(q_0,w) = \hat{\gamma}(q_0,w)$, tj. $w \in L(\mathcal{M}) \iff w \in L(\mathcal{M}')$. Případ $w = \varepsilon$ je ošetřen v definici koncových stavů G takto: $\varepsilon \in L(\mathcal{M}) \iff D_\varepsilon(q_0) \cap F \neq \emptyset \iff q_0 \in G \iff \varepsilon \in L(\mathcal{M}')$. Tedy celkem dostáváme $L(\mathcal{M}) = L(\mathcal{M}')$. □

Předchozí věta podává algoritmus pro transformaci automatu s ε-kroky na ekvivalentní nedeterministický automat (bez ε-kroků). Pokud jej aplikujeme na automat z příkladu 2.45, pak dostaneme



2.2.3 Uzávěrové vlastnosti regulárních jazyků – část I

V této části dokážeme, že třída regulárních jazyků je uzavřená na některé operace nad jazyky definované v části 1.1.1. Důležitost uzávěrových vlastností spočívá nejen ve zkoumání vlastností dané třídy jazyků, ale má i řadu praktických aspektů. Daný jazyk lze často vyjádřit pomocí několika jazyků jednodušších, pro každý z nich navrhnout konečný automat a na jejich základě sestrojit žádaný výsledný automat – tento postup byl již ilustrován v příkladu 2.12. Uzávěrových vlastností lze často využít i ke zjednodušení důkazu, že nějaký jazyk je (viz opět 2.12) či (a to zejména) není regulární – viz příklad 2.50 níže.

Uvědomme si, že již na počátku této kapitoly (přesněji v 2.9 – 2.11) jsme de facto ukázali, že třída regulárních jazyků je uzavřená na základní množinové operace sjednocení, průniku a komplementu.

Věta 2.49. *Třída regulárních jazyků je uzavřená na sjednocení, průnik a rozdíl.*

Důkaz. Nechť L_1 a L_2 jsou regulární jazyky rozpoznávané deterministickými konečnými automaty $\mathcal{M}_1$ a $\mathcal{M}_2$ s totálními přechodovými funkcemi. Bez újmy na obecnosti přitom můžeme předpokládat, že L_1 i L_2 jsou jazyky nad stejnou abecedou Σ (v opačném případě provedeme sjednocení abeced a dodefinujeme přechodové funkce automatů – viz lemma 2.6). Pak jazyky $L_1 \cup L_2$, $L_1 \cap L_2$ a $L_1 - L_2$ jsou rozpoznávané po řadě automaty $\mathcal{M}_1 \Cup \mathcal{M}_2$, $\mathcal{M}_1 \Cap \mathcal{M}_2$ a $\mathcal{M}_1 \ominus \mathcal{M}_2$ (viz definice 2.9 a poznámka 2.11), jsou tedy regulární. □

Příklad 2.50. *Ukažme, že jazyk $L = \{w \in \{a,b\}^* \mid \#_a(w) = \#_b(w)\}$ není regulární. Důkaz lze samozřejmě provést pomocí Myhillovy-Nerodovy věty (viz 2.28) či lemmatu o vkládání 2.13. Jestliže však již víme, že $L_1 = \{a^i b^i \mid i \geq 0\}$ není regulární, stačí uvážit, že $L_1 = L \cap a^* b^*$. Kdyby totiž byl L regulární, pak by díky regularitě $a^* b^*$ a uzavřenosti regulárních jazyků na průnik musel být regulární i L_1, což ale není.*

Věta 2.51. *Třída regulárních jazyků je uzavřená na komplement.*

Důkaz. Důkaz okamžitě plyne z předchozí věty 2.49, neboť co–$L = \Sigma^* - L$. Lze též uvést konstruktivní důkaz: nechť L je regulární jazyk nad abecedou Σ, rozpoznávaný deterministickým konečným automatem $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ s totální přechodovou funkcí. Pak jazyk co–L je rozpoznávaný konečným automatem $\overline{M} = (Q, \Sigma, \delta, q_0, Q - F)$ (viz poznámka 2.11). □

Věta 2.52. *Třída regulárních jazyků je uzavřená na zřetězení.*

Důkaz. Nechť L_1 a L_2 jsou regulární jazyky, rozpoznávané nedeterministickými konečnými automaty $\mathcal{M}_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ a $\mathcal{M}_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$, kde $Q_1 \cap Q_2 = \emptyset$ (předpoklad, že oba automaty jsou nedeterministické, není podstatný; umožní nám však elegantně zapsat definici přechodové funkce níže uvedeného automatu). Pak $L_1.L_2$ je rozpoznávaný automatem $\mathcal{M}_1 \odot \mathcal{M}_2 = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, q_1, F_2)$ s ε-kroky, kde

$$\delta = \delta_1 \cup \delta_2 \cup \{((p, \varepsilon), \{q_2\}) \mid p \in F_1\}$$

Jinými slovy, automat $\mathcal{M}_1 \odot \mathcal{M}_2$ vznikne „napojením" automatů $\mathcal{M}_1$ a $\mathcal{M}_2$ pomocí ε-hran, které vedou z koncových stavů $\mathcal{M}_1$ do počátečního stavu $\mathcal{M}_2$. Zřejmě $L(\mathcal{M}_1 \odot \mathcal{M}_2) = L_1.L_2$, neboť $w \in L(\mathcal{M}_1 \odot \mathcal{M}_2) \iff w = uv$, kde $u \in L(\mathcal{M}_1)$ a $v \in L(\mathcal{M}_2) \iff uv \in L_1.L_2$. □

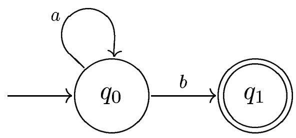
Věta 2.53. *Třída regulárních jazyků je uzavřená na iteraci.*

Důkaz. Nechť L je regulární jazyk, rozpoznávaný nedeterministickým konečným automatem $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ (předpoklad, že $\mathcal{M}$ je nedeterministický, má stejný význam jako v důkazu předchozí věty). Pak jazyk L^* je rozpoznávaný automatem $\mathcal{M}_* = (Q \cup \{p\}, \Sigma, \delta', p, \{p\})$ s ε-kroky, kde $p \notin Q$ a

$$\delta' = \delta \cup \{((p, \varepsilon), \{q_0\})\} \cup \{((q, \varepsilon), \{p\}) \mid q \in F\}$$

V automatu $\mathcal{M}_*$ tedy přibyl nový počáteční stav p, který je také jediným koncovým stavem, dále ε-hrana ze stavu p do původního počátečního stavu q_0 a konečně ε-hrany z původních koncových stavů do stavu p. Zřejmě $L(\mathcal{M}_*) = L^*$. □

Poznámka 2.54. *Zavedení nového počátečního stavu p v automatu $\mathcal{M}'$ z předchozího důkazu je nutné – kdybychom tak neučinili a prohlásili za koncový přímo stav q_0 (počáteční stav automatu který rozpoznává L^* musí být nutně koncový, neboť L^* vždy obsahuje prázdné slovo), do kterého bychom přidali ε-hrany z původních koncových stavů, mohl by vzniklý automat přijímat vlastní nadmnožinu jazyka L^* – dojít k tomu může tehdy, když v $\mathcal{M}$ existuje alespoň jeden přechod do stavu q_0 a přitom q_0 není v automatu $\mathcal{M}$ koncovým stavem. Ukážeme si konkrétní příklad – nechť $\mathcal{M}$ je automat:*



Kdybychom nyní přidali ε-hranu ze stavu q_1 do q_0 a stav q_0 prohlásili za koncový, bude vzniklý automat akceptovat např. i slovo aa, které do iterace jazyka $L(\mathcal{M})$ nepatří.

Věta 2.55. *Třída regulárních jazyků je uzavřena vůči operaci zrcadlového obrazu (reverzi).*

Důkaz. Nechť L je regulární jazyk, který je rozpoznáván nějakým konečným automatem $\mathcal{M}$ a sestrojme s ním ekvivaletní NFA $\mathcal{M}'$ s ε-kroky. $\mathcal{M}'$ obržíme tak, že (neformálně řečeno) v přechodovém grafu $\mathcal{M}$ obrátíme orientaci hran, přidáme nový počáteční stav, z něhož povedeme ε-přechody do všech koncových stavů automatu $\mathcal{M}$ a počáteční stav $\mathcal{M}$ bude koncovým stavem v $\mathcal{M}'$. Formální definici $\mathcal{M}'$ a důkaz ekvivalence $L(\mathcal{M}) = L(\mathcal{M}')$ ponecháváme čtenáři. □

Důsledek 2.56. *Libovolný jazyk L je regulární $\iff$ L^R je regulární.*

Důkaz. Díky 2.55 zbývá ukázat implikaci L^R regulární $\implies L$ je regulární. Jelikož platí $(L^R)^R = L$, pak dle 2.55 dostáváme L^R regulární $\implies (L^R)^R = L$ je regulární. □

Uzavřenost na další operace bude ukázána v části 2.3.1, kde s výhodou využijeme dalšího konservativního rozšíření základního modelu.

2.2.4 Regulární výrazy

V této části zavedeme formalismus, který je rovněž užitečným prostředkem pro popis (specifikaci) a návrh konečných automatů. Jen zopakujme, že zatím jsme pojem regulární jazyk používali jako (syntaktickou) zkratku pro pojem jazyk přijímaný konečným automatem a též pro pojem jazyk generovaný gramatikou typu 3. Tvrzení, že se jedná o tutéž třídu jazyků bude ukázáno v následující části. V této části budeme definovat regulární jazyky tak, jak byly historicky originálně (pod tímto názvem) zavedeny, uvedeme formalismus pro jejich specifikaci a konečně ukážeme, že takto definovaná třída jazyků je identická s třídou jazyků rozpoznávaných konečnými automaty.

Definice 2.57. Třída regulárních jazyků nad abecedou Σ, označovaná jako $R(\Sigma)$, je definována induktivně takto:

1. $\emptyset, \{\varepsilon\}$ a $\{a\}$ pro každé $a \in \Sigma$ je regulární jazyk nad Σ.
2. Jsou-li L_1, L_2 regulární jazyky nad Σ, jsou také, $L_1.L_2, L_1 \cup L_2$ a L_1^* regulární jazyky[4] nad Σ.
3. Každý regulární jazyk vznikne po konečném počtu aplikací kroků 1 a 2.

Jinými slovy $R(\Sigma)$ je nejmenší třída jazyků nad Σ splňující podmínky 1 a 2. Jazyky uvedené ad 1 se nazývají elementární, operace nad jazyky uvedené ad 2 se nazývají regulární. Je tedy vidět, že každý regulární jazyk lze popsat určením elementárních jazyků a předpisu, který určuje jak na tyto jazyky aplikovat regulární operace. Taková specifikace je cílem následující definice.

Definice 2.58. Množina *regulárních výrazů* (regular expressions) nad abecedou Σ, označovaná *RE*(Σ), je definována induktivně takto:

1. $\boldsymbol{\varepsilon}, \boldsymbol{\emptyset}$ a $\boldsymbol{a}$ pro každé $a \in \Sigma$ jsou regulární výrazy nad Σ (tzv. *základní* regulární výrazy).
2. Jsou-li E, F regulární výrazy nad Σ, pak také $(E.F), (E + F)$ a (E^*) jsou regulární výrazy nad Σ.
3. Každý regulární výraz vznikne po konečném počtu aplikací kroků 1–2.

Základní regulární výrazy se podobají symbolům, se kterými jsme se v textu běžně setkávali. Tučně jsou zapsány proto, že je třeba je chápat jako symboly zcela nové; tyto „dvojníky“ jsme zavedli proto, abychom mohli vždy snadno rozlišit mezi *syntaxí* a *sémantikou* regulárních výrazů (předchozí definice popisuje pouze syntaxi). V regulárních výrazech se mohou vyskytovat také kulaté závorky jako metasymboly, které pomáhají vymezit rozsah operátorů. Abychom jejich použití omezili na minimum, přijmeme konvenci týkající se priority operátorů: Největší prioritu má „$*$“, pak „.“ a nakonec „$+$“, přičemž „nadbytečné“ závorky lze vypouštět. Výraz $\boldsymbol{a} + \boldsymbol{b}.\boldsymbol{c}^*$ tedy odpovídá výrazu $(\boldsymbol{a} + (\boldsymbol{b}.(\boldsymbol{c})^*))$.

Každý regulární výraz E nad abecedou Σ *popisuje* (jednoznačně určuje) jazyk $L(E)$

4. je zřejmé, že požadavek „$\{\varepsilon\}$ je regulární …“ v 1 je nadbytečný, protože $\{\varepsilon\} = \emptyset^*$. Je uveden čistě z pedagogických důvodů.

nad abecedou Σ (jazyk $L(E)$ je *sémantikou* regulárního výrazu E) podle těchto pravidel:

$$
\begin{aligned}
L(\boldsymbol{\varepsilon}) &\stackrel{def}{=} \{\varepsilon\} \\
L(\boldsymbol{\emptyset}) &\stackrel{def}{=} \emptyset \\
L(\boldsymbol{a}) &\stackrel{def}{=} \{a\} \text{ pro každé } a \in \Sigma \\
L(E.F) &\stackrel{def}{=} L(E).L(F) \\
L(E+F) &\stackrel{def}{=} L(E) \cup L(F) \\
L(E^*) &\stackrel{def}{=} L(E)^*
\end{aligned}
$$

Na levé straně definujících rovnic se vyskytují regulární výrazy; na pravé straně je třeba symboly „ε“, „$\emptyset$“, atd. chápat v jejich původním významu – tj. jazyk určený regulárním výrazem $\boldsymbol{\varepsilon}$ obsahuje pouze prázdné slovo, jazyk určený výrazem $\boldsymbol{\emptyset}$ je prázdný, atd. Tečka se v zápisu regulárních výrazů často vynechává. Například tedy platí:

$$
\begin{aligned}
L((\boldsymbol{a}+\boldsymbol{b})^*(\boldsymbol{ab}+\boldsymbol{bb})(\boldsymbol{a}+\boldsymbol{b})^*) &= \{w \in (a,b)^* \mid w \text{ obsahuje podslovo } ab \text{ nebo } bb\} \\
L((\boldsymbol{aa}+\boldsymbol{ab}+\boldsymbol{ba}+\boldsymbol{bb})^*) &= \{w \in \{a,b\}^* \mid w \text{ má sudou délku}\} \\
L((\mathbf{0}^*\mathbf{1}^*\mathbf{2}^*)^*) &= \{0,1,2\}^*
\end{aligned}
$$

Poznámka 2.59. *Z výše řečeného se okamžitě nahlédne fakt, že jazyk je regulární nad* Σ*, právě když jej popisuje nějaký regulární výraz nad* Σ*. V tomto případě též říkáme, že jazyk* je popsatelný *nějakým regulárním výrazem.*

Na rozdíl od regulárních gramatik neobsahují regulární výrazy žádnou formu rekurze; aby bylo jasné, oč se jedná, uvažme např. gramatiku $\mathcal{G} = (\{S,A\},\{a,b\},P,S)$, kde P obsahuje tato pravidla:

$S \to aA \mid a$,
$A \to bS \mid b$.

Neterminály S a A přirozeným způsobem určují jazyky $L(S)$ a $L(A)$:

$$
\begin{aligned}
L(S) &= \{w \in \{a,b\}^* \mid S \Rightarrow^* w\} \\
L(A) &= \{w \in \{a,b\}^* \mid A \Rightarrow^* w\}
\end{aligned}
$$

Platí tedy $L(\mathcal{G}) = L(S)$. Pravidla gramatiky $\mathcal{G}$ lze přirozeným způsobem transformovat na systém vzájemně rekurzivních rovnic:

$$
\begin{aligned}
X_S &= \{a\}.X_A \cup \{a\} \\
X_A &= \{b\}.X_S \cup \{b\}
\end{aligned}
$$

Proměnné X_S a X_A zastupují jazyky nad abecedou $\{a,b\}$. První rovnice říká, že jazyk X_S dostaneme sjednocením zřetězení jazyků $\{a\}$ a X_A s jazykem $\{a\}$. Význam druhé rovnice je obdobný – jazyk X_A je definován v závislosti na jazyku X_S.

Dosadíme-li za X_S jazyk $L(S)$ a za X_A jazyk $L(A)$, obě rovnice platí. Není obtížné dokázat (čtenář se o to může pokusit), že $L(S)$ a $L(A)$ jsou také *jediným* řešením uvedených rovnic. Tento fakt má obecnou platnost, tj. jazyk generovaný *libovolnou* regulární gramatikou $\mathcal{G}$ lze tímto způsobem vyjádřit jako řešení jistého systému rekurzivních rovnic. Podobná forma rekurze se objevuje také u konečných automatů; v definici 2.32 jsme pro každý stav q konečného automatu $\mathcal{M}$ zavedli jazyk $L(q)$. Přechodová funkce popisuje závislost mezi jazyky $L(q)$ podobným způsobem, jako množina pravidel v případě gramatik.

Rekurze je z hlediska popisné síly velmi důležitá – pokud definující rovnice určené pravidly regulární gramatiky $\mathcal{G}$ neobsahují žádnou rekurzivní (tj. „cyklickou“) závislost mezi proměnnými, je jazyk $L(\mathcal{G})$ nutně *konečný*. Příkladem takové gramatiky je třeba $(\{S, A, B\}, \{a, b, c\}, P, S)$, kde P obsahuje pravidla

$S \to a \mid bA$
$A \to c \mid aB$
$B \to b \mid c$

V případě regulárních výrazů, kde se rekurze v žádné podobě neobjevuje, se snadno vidí, že jediný prostředek umožňující definovat nekonečný jazyk je *iterace*. Výsledek o ekvivalenci regulárních výrazů a regulárních gramatik, který v této části dokážeme, lze tedy také interpretovat jako ekvivalenci popisné síly iterace a rekurze ve speciálním typu rovnic.

Věta 2.60. *Nechť E je regulární výraz. Pak existuje konečný automat rozpoznávající $L(E)$.*

Důkaz. Pro libovolnou (konečnou) abecedu Σ lze zkonstruovat FA, které rozpoznávají elementární jazyky, tj. $\emptyset, \{\varepsilon\}$ a $\{a\}$ pro každé $a \in \Sigma$. Tvrzení věty pak okamžitě plyne z uzavřenosti jazyků rozpoznatelných konečnými automaty vůči regulárním operacím, tj. sjednocení (viz věta 2.49), zřetězení(viz věta 2.52) a iteraci (viz věta 2.53). □

Čtenáře, který očekával přímočařejší algoritmus, jak k danému regulárnímu výrazu sestrojit ekvivalentní konečný automat, nezklameme a odkazujeme jej na pojem regulárního přechodového grafu a větu 2.65, které jsou uvedeny v závěru této části.

Poznámka 2.61. *Výše uvedená věta 2.60 tedy říká, že třída jazyků, která obsahuje všechny konečné množiny (každá z nich jistě rozpoznatelná nějakým FA) a která je uzavřena na sjednocení, zřetězení a iteraci je podtřídou třídy jazyků rozpoznatelných konečnými automaty. Následující věta ukazuje, že platí i obrácená inkluse, což spolu s 2.60 říká, že* třída jazyků rozpoznatelných konečnými automaty je nejmenší třída, která obsahuje všechny konečné množiny a je uzavřena na sjednocení, zřetězení a iteraci – *srovnej s formulací tzv. Kleeneho věty (věta 2.63).*

Věta 2.62. *Nechť L je akceptovaný nějakým (libovolným) DFA, pak L je popsatelný nějakým regulárním výrazem.*

Důkaz. Nechť $\mathcal{A} = (\{q_1, \ldots, q_n\}, \Sigma, \delta, q_1, F)$ je DFA, který akceptuje L. Pro všechna $i, j : 1 \leq i, j \leq n$ definujme množiny slov

$R_{ij} \stackrel{def}{=} \{w \in \Sigma^* \mid \hat{\delta}(q_i, w) = q_j\}$

tj. množiny slov převádějících $\mathcal{A}$ ze stavu q_i do stavu q_j. Je vhodné si uvědomit, že platí:

$$L = L(\mathcal{A}) = \bigcup_{q_j \in F} R_{1j} \qquad (*)$$

Stačí tedy ukázat, že každá R_{1j} je regulární (definice regulárního jazyka v sobě obsahuje uzavřenost na sjednocení), a tedy je popsatelná nějakým regulárním výrazem.

Abychom dokázali regularitu R_{1j}, dokažme, že každá R_{ij} je regulární (žádané dostaneme specializací pro $i = 1$). Za tímto účelem definujme množiny slov $R^k_{ij}, k = 0, \ldots, n$, kde každá z nich bude množinou všech slov, která převádějí automat $\mathcal{A}$ ze stavu q_i do stavu q_j (jako u R_{ij}), ale při tomto výpočtu není povoleno projít žádným (mezi)stavem q_m, který by měl index m větší než k. Formálně zapsáno:

$$R^k_{ij} \overset{def}{=} \{x \mid \hat{\delta}(q_i, x) = q_j \wedge ((\hat{\delta}(q_i, y) = q_m \wedge \varepsilon \prec y \prec x) \Rightarrow m \leq k)\}$$

Zřejmě pak platí $R_{ij} = R^n_{ij}$ (a ukážeme-li regularitu všech R^k_{ij}, stačí položit $k = n$).

Nyní si uvědomme, že R^k_{ij} lze definovat induktivně, a tedy následně využít k dokazování regularity R^k_{ij} indukci. Navíc můžeme takovou definici využít k jejich výpočtu (ať už k rekursivnímu či iterativnímu). Definujme tedy:

1. $R^0_{ij} \overset{def}{=} \begin{cases} \{a \in \Sigma \mid \delta(q_i, a) = q_j\} & \text{je-li } i \neq j \\ \{a \in \Sigma \mid \delta(q_i, a) = q_j\} \cup \{\varepsilon\} & \text{je-li } i = j \end{cases}$

2. $R^{k+1}_{ij} \overset{def}{=} R^k_{i,k+1}(R^k_{k+1,k+1})^* R^k_{k+1,j} \cup R^k_{ij}$ pro všechna $k = 0, \ldots, n-1$ (**)

Neformálně lze tuto definici vysvětlit takto: bod 1 představuje bázi výše uvedené definice. Bod 2 říká, že vstup, který automat $\mathcal{A}$ převede z q_i do q_j bez průchodu mezistavem o indexu větším než $k + 1$, je buď

- z množiny R^k_{ij} (2. sčítanec v 2), tj. množiny slov odpovídající cestám z q_i do q_j, kdy není použit žádný mezistav „vyšší“ než q_k, nebo
- z množiny slov, která reprezentuje průchod stavem „vyšším“ než q_k, tj. stavem q_{k+1} (viz 1. sčítanec v 2) – tuto množinu slov lze získat tak, že
  1. nejprve vezmeme všechna ta slova, která automat $\mathcal{A}$ převedou z q_i *poprvé* do q_{k+1} (tj. 1. činitel $R^k_{i,k+1}$),
  2. následovaná (zřetězená se) všemi slovy, která převedou $\mathcal{A}$ z q_{k+1} zpět do q_{k+1}, aniž by se prošlo nějakým vyšším stavem než q_k. Pokud je množina těchto slov neprázdná, pak reprezentuje cyklus z q_{k+1} zpět do q_{k+1}, a proto tedy u 2. činitele $(R^k_{k+1,k+1})^*$ je operace iterace. Množina těchto slov může být i prázdná (taková cesta neexistuje) – uvědomme si, že definice je korektní i v tomto případě, protože $\emptyset^* = \{\varepsilon\}$.
  3. Konečně ke slovům získaným ve dvou výše uvedených bodech připojíme (zřetězením) všechna slova, která převedou $\mathcal{A}$ z q_{k+1} do cílového q_j bez průchodu stavem q_{k+1} nebo vyšším, což je reprezentováno 3. činitelem $R^k_{k+1,j}$.

Nyní se indukcí snadno ukáže, že každá R^k_{ij} je regulární, a tedy popsatená nějakým regulárním výrazem. I když z hlediska důkazu není nutno tyto výrazy specifikovat, uvedeme je (v komentářových závorkách '(*' a '*)'), neboť tak de facto specifikujeme algoritmus pro konstrukci hledaného výrazu.

Báze indukce, $k = 0$. Pro libovolná i, j je $R^0_{ij} \subseteq \Sigma \cup \{\varepsilon\}$, a tedy regulární – obsahuje totiž jen elementární jazyky.

(*Odpovídající regulární výraz, značený r^0_{ij}, lze zapsat jako $a_1 + \ldots + a_p$ (resp. $a_1 + \ldots + a_p + \varepsilon$, pokud $i = j$), kde $\{a_1, \ldots, a_p\}$ je množina všech symbolů a takových, že $\delta(q_i, a) = q_j$. Jetliže žádné takové a neexistuje, pak $r^0_{ij} = \emptyset$ (resp. $r^0_{ij} = \varepsilon$, pokud $i = j$).*)

Indukční krok. (IP): předpokládejme , že pro jisté $k, 0 \leq k < n$ a všechna i, j jsou R^k_{ij} regulární a ukažme, že pak i R^{k+1}_{ij} je regulární. To se však snadno nahlédne: R^{k+1}_{ij} je definována – viz (**) – pomocí R^k_{ij} (dle IP jsou regulární) a pomocí regulárních operací sjednocení, zřetězení a iterace, a tedy (viz definice regulárního jazyka) je regulární i R^{k+1}_{ij}. (*IP je ekvivalentní tomu, že pro všechna l, m existují regulární výrazy r^k_{lm} takové, že $L(r^k_{lm}) = R^k_{lm}$. Hledaný výraz r^{k+1}_{ij} je tedy roven $r^k_{i,k+1}(r^k_{k+1,k+1})^* r^k_{k+1,j} \cup r^k_{ij}$.*)

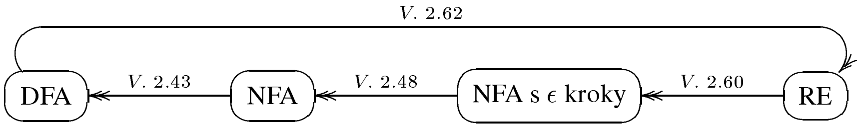
Tímto je důkaz ukončen (zopakujme: všechna R^k_{ij} jsou regulární, specielně R^n_{ij} jsou regulární a $R^n_{ij} = R_{ij}$, a tedy i R_{1j} je regulární a tedy dle (*) $L(\mathcal{A}) = \bigcup_{q_j \in F} R_{1j}$ je regulární). (*Tedy $L(\mathcal{A})$ je popisován výrazem $r^n_{1j_1} + \ldots + r^n_{1j_p}$, kde $F = \{q_{j_1}, \ldots q_{j_p}\}$.*)

□

Obě předchozí věty 2.60 a 2.62 lze shrnout do tvrzení známého jako Kleeneho věta:

Věta 2.63. ***(Kleene).*** *Libovolný jazyk je popsatelný regulárním výrazem právě když je rozpoznatelný konečným automatem.*

V této části jsme dosud zavedli jistá rozšíření deterministických konečných automatů definovaných v části 2.1 a ukázali jejich vzájemnou ekvivalenci. Strukturu těchto výsledků (mimo průběžně uváděné uzávěrové vlastnosti) lze znázornit takto:



Důkaz věty 2.62 – viz definice (**) – podává i návod, jak napsat algoritmus převádějící libovolný konečný automat na ekvivaletní regulární výraz.

1. Zmíněnou definici lze považovat za specifikaci rekursivní procedury `P(i,j,k)`, která má (při značení jako v důkazu věty 2.62) vypočítat r^k_{ij}, přičemž rekursivní smyčka končí pro $k = 0$. Uvědomme si však, že takový algoritmus by byl značně neefektivní - pro řadu konkrétních hodnot parametrů i, j, k by (v obecném případě) opakovaně počítal již v některém předchozím rekursivní volání vypočtené r^k_{ij}.

2. Výše uvedenou neefektivitu lze odstranit (na úkor nárůstu paměťových nároků) tak, že bychom navíc deklarovali pole `B[1..n,1..n,1..n]` a `V[1..n,1..n,1..n]`, kde bychom inicializovali `B[i,j,k]=0` pro všechna i, j, k a kde nastavení `B[i,j,k]=1` by udávalo, že výraz r^k_{ij} již byl (některou z předchozích aktivací procedury `P(i,j,k)`) vypočítán a jeho hodnota by byla uložena ve `V[i,j,k]`.

3. Dále si uvědomme, že výše uvedenou rekursi lze v tomto konkrétním případě, převést na iteraci. Její inicializace spočívá ve výpočtu r^0_{ij} pro všechna i, j. Vlastní iterace postupně počítá (opět pro všechna i, j) výrazy $r^1_{ij}, \ldots, r^n_{ij}$. Další možné vylepšení (týkající se nyní nároků paměťových) spočívá v uvědomění si, že k výpočtu r^{k+1}_{ij} nepotřebujeme znát všechny dosud určené hodnoty $r^m_{ij}, 0 \leq m \leq k$, ale stačí uchovávat jen hodnoty r^k_{ij}.

Implementační detaily ponecháváme čtenáři s tím, že je vhodné si uvědomit, že pracnost implementace (i v tomto jednoduchém případě) je zřejmě nejnižší ad 1 (lze ji uvažovat jako vhodný prototyp) a nejvyšší ad 3. Na druhé straně asi ani jeden z nastíněných algoritmů není vhodný pro „ruční" zpracování (stylem papír a tužka). Uvedeme proto ještě další nástroj, který umožní „ruční" převody od konečného automatu k regulárnímu výrazu a obráceně. Mělo by býti zřejmé (nicméně bude dokázáno), že se opět jedná o konservativní rozšíření modelu konečného automatu.

Definice 2.64. *Regulární přechodový graf* $\mathcal{M}$ je pětice $(Q, \Sigma, \delta, I, F)$, kde

- Q je neprázdná konečná množina stavů.
- Σ je vstupní abeceda.
- $\delta : Q \times Q \to RE(\Sigma)$ je parciální přechodová funkce.
- $I \subseteq Q$ je množina počátečních stavů.
- $F \subseteq Q$ je množina koncových stavů.

Regulární přechodové grafy tedy představují další zobecnění automatů s ε-kroky. Hrany mohou být nyní ohodnoceny nejen prvky $\Sigma \cup \{\varepsilon\}$, ale dokonce libovolným regulárním výrazem nad Σ (mezi libovolnými dvěma uzly je však nejvýše jedna hrana, což však není omezení podstatné – viz operátor +). Každý automat s ε-kroky lze považovat za regulární přechodový graf s jedním počátečním stavem, jehož hrany jsou ohodnoceny regulárními výrazy tvaru $\boldsymbol{a_1} + \cdots + \boldsymbol{a_n}$, kde $n \in \mathbb{N}$ a každé $\boldsymbol{a_i}$ patří do $\Sigma \cup \{\boldsymbol{\varepsilon}\}$.

Slovo $w \in \Sigma^*$ je grafem $\mathcal{M}$ *akceptováno*, právě když existuje posloupnost stavů $q_0, \ldots, q_n$, kde $n \geq 1$, $q_0 \in I$, $q_n \in F$ a $\delta(q_{i-1}, q_i)$ je definováno pro každé $0 < i \leq n$ taková, že w lze rozdělit na n částí $w = v_1 \ldots v_n$ tak, že $v_i \in L(\delta(q_{i-1}, q_i))$ pro každé $0 < i \leq n$. Navíc ε je akceptováno také tehdy, je-li $I \cap F \neq \emptyset$.

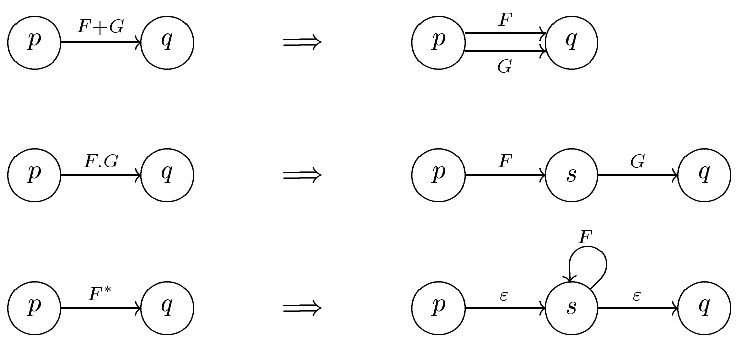
Věta 2.65. *Pro libovolný regulární přechodový graf $\mathcal{M} = (Q, \Sigma, \delta, I, F)$ existuje ekvivalentní NFA $\mathcal{M}'$ s ε-kroky.*

Důkaz. Přechodový graf automatu $\mathcal{M}'$ s ε-kroky vznikne transformací regulárního přechodového grafu $\mathcal{M}$ (tento důkazový postup je korektní, neboť každý automat s ε-kroky lze jednoznačně reprezentovat jeho přechodovým grafem a obráceně). Nejprve přidáme ke grafu $\mathcal{M}$ nový stav q_0 a hranu $q_0 \xrightarrow{\varepsilon} q$ pro každé $q \in I$. Stav q_0 bude (jediným) počátečním stavem automatu $\mathcal{M}'$, prvky F jeho koncovými stavy. Další postup transformace spočívá v opakované realizaci dvou kroků (viz obrázek 2.3):

1. Odstraň všechny hrany, které jsou ohodnoceny symbolem $\emptyset$.
2. Vyber libovolnou hranu $p \xrightarrow{E} q$, kde $E \notin \Sigma \cup \{\varepsilon\}$, odstraň ji a proveď následující:
   - Pokud $E = F + G$, přidej hrany $p \xrightarrow{F} q$ a $p \xrightarrow{G} q$.
   - Pokud $E = F.G$, přidej ke Q nový stav s a hrany $p \xrightarrow{F} s$ a $s \xrightarrow{G} q$.
   - Pokud $E = F^*$, přidej ke Q nový stav s a hrany $p \xrightarrow{\varepsilon} s$, $s \xrightarrow{F} s$ a $s \xrightarrow{\varepsilon} q$.

Tyto dva kroky se provádí tak dlouho, dokud přechodový graf obsahuje alespoň jednu hranu ohodnocenou symbolem, který nepatří do $\Sigma \cup \{\varepsilon\}$. Fakt, že popsaná transformace skončí, vyplývá z toho, že $\mathcal{M}$ obsahuje pouze konečně mnoho hran a každý regulární výraz vznikne aplikací pravidel 1–2 z definice 2.58 v konečně mnoha krocích. Výsledný graf je přechodovým grafem automatu $\mathcal{M}'$ s ε-kroky. Snadno se ověří, že kroky 1 a 2 popsaného transformačního algoritmu zachovávají ekvivalenci automatů, proto $L(\mathcal{M}) = L(\mathcal{M}')$. □

Věta 2.66. *Pro každý regulární přechodový graf $\mathcal{M} = (Q, \Sigma, \delta, I, F)$ existuje ekvivalentní regulární přechodový graf $\mathcal{M}' = (\{x, y\}, \Sigma, \delta, \{x\}, \{y\})$, kde δ je definováno pouze pro dvojici (x, y).*



Obrázek 2.3: Pravidla pro transformaci regulárního přechodového grafu na ekvivalentní NFA s ε-kroky

Důkaz. Popíšeme způsob, kterým lze transformovat graf $\mathcal{M}$ na graf $\mathcal{M}'$. Nejprve přidáme ke grafu $\mathcal{M}$ nový počáteční stav x a nový koncový stav y. Přidáme také hrany $x \stackrel{\varepsilon}{\to} q$ pro každé $q \in I$ a $r \stackrel{\varepsilon}{\to} y$ pro každé $r \in F$. Tato úprava jistě nezmění přijímaný jazyk. Každý stav p různý od x a y nyní odstraníme spolu s jeho incidentními hranami (tj. hranami, které do p vcházejí nebo z p vycházejí) takto (viz obrázek 2.4): Nechť $\{E_1, \ldots, E_n\}$ je množina všech regulárních výrazů E takových, že $p \stackrel{E}{\to} p$ je hrana v upravovaném přechodovém grafu. Pro každou dvojici hran $r \stackrel{F}{\to} p$, kde $r \neq p$ a $p \stackrel{G}{\to} q$, kde $p \neq q$, přidáme hranu z r do q s ohodnocením $F.(E_1 + \cdots + E_n + \emptyset)^*.G$. Pak lze stav p spolu s incidentními hranami odstranit bez toho, aby se změnil přijímaný jazyk ($\emptyset$ bylo k množině $\{E_1, \ldots, E_n\}$ přidáno pro případ, že uvedená množina je prázdná; připomeňme $\emptyset^* = \{\varepsilon\}$). Pokud má stav p tu vlastnost, že do něj žádná hrana nevchází, resp. z něj žádná hrana nevychází, je p z počátečního stavu x nedosažitelný, resp. nelze z něj dosáhnout koncového stavu y. V takovém případě můžeme p odstranit aniž bychom nějaké hrany přidávali – tato úprava přijímaný jazyk nezmění.

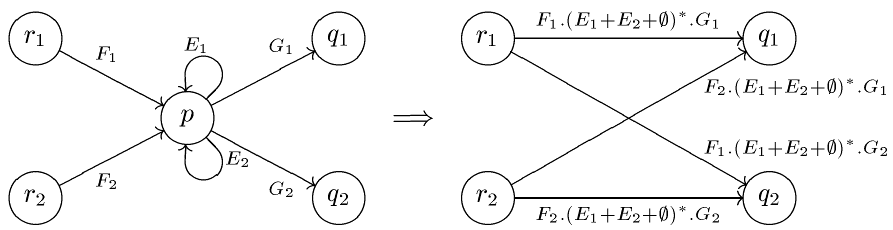
Po odstranění všech stavů různých od x a y zůstanou tyto dva stavy spolu s konečně mnoha hranami z x do y (hrana z x do x, nebo z y do y pomocí výše uvedené transformace vzniknout nemůže). Nechť $\{E_1, \ldots, E_m\}$ je množina všech regulárních výrazů, které se na těchto hranách vyskytují. Všechny hrany pak můžeme odstranit a přidat jedinou hranu s ohodnocením $E_1 + \cdots + E_m + \emptyset$. Tato úprava přijímaný jazyk rovněž nezmění. Obdržíme tak přechodový graf $\mathcal{M}'$ se dvěma stavy a jedinou hranou. $\square$

Shrnutí obou předchozích vět by bylo jen opakováním Kleeneho věty – viz 2.63.

2.3 Vlastnosti regulárních jazyků

2.3.1 Uzávěrové vlastnosti regulárních jazyků – část II

Již dříve jsme ukázali, že třída regulární jazyků je uzavřena vůči sjednocení, průniku, komplementu, zřetězení a iteraci (viz část 2.2.3). Díky Kleeneho větě lze velmi snadno ukázat, že třída regulárních jazyků je uzavřena vůči operaci (regulární) substituce.



Obrázek 2.4: Eliminace stavu regulárního přechodového grafu.

Věta 2.67. *Nechť Σ je konečná abeceda a $f : \Sigma \to \Delta$ substituce taková, že $f(a)$ je regulární jazyk nad Δ pro každé $a \in \Sigma$. Pak pro každý regulární jazyk L je též $f(L)$ regulárním jazykem.*

Důkaz. Nechť E je regulární výraz takový, že $L(E) = L$ a E_a jsou regulární výrazy takové, že $L(E_a) = f(a)$ pro každé $a \in \Sigma$. Pokud pro každé a dosadíme do E za každý výskyt symbolu a odpovídající regulární výraz E_a, získáme opět regulární výraz, řekněmě F, a to takový, že $L(F) = f(L)$. Formální důkaz korektnosti těchto dosazení lze provést indukcí vzhledem k počtu operátorů v regulárním výrazu, přičemž bychom vzali v potaz, že $f(L_1 \cup L_2) = f(L_1) \cup f(L_2)$ a analogicky pro zřetězení a iteraci. □

Věta 2.68. *Třída regulárních jazyků je uzavřena vůči homomorfismu a inversnímu homomorfismu.*

Důkaz. Uzavřenost vůči homomorfismu okamžitě plyne z uzavřenosti vůči substituci: každý homomorfismus h je substitucí, kde každé $h(a), a \in \Sigma$, je jednoprvkový jazyk.

Abychom ukázali uzavřenost vůči inversnímu homomorfismu, mějme DFA $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ takový, že $L = L(\mathcal{M})$ a homomorfismus h z Δ do Σ^*. Zkonstruujme DFA $\mathcal{M}'$ akceptující $h^{-1}(L)$ tak, že čte symbol $a \in \Delta$ a simuluje činnost $\mathcal{M}$ nad $h(a)$. Formálně $\mathcal{M}' = (Q, \Delta, \delta', q_0, F)$, kde pro všechna $a \in \Delta$ a $q \in Q$ definujeme $\delta'(q, a) = \hat{\delta}(q, h(a))$. Poznamenejme, že $h(a)$ je slovo (dokonce může být i prázdné), avšak $\hat{\delta}$ jako rozšířená přechodová funkce je definována pro všechna slova. Indukcí vzhledem k délce slova x lze snadno ukázat, že $\hat{\delta}'(q_0, x) = \hat{\delta}(q_0, h(x))$, a tedy $\mathcal{M}'$ akceptuje x právě když $\mathcal{M}$ akceptuje $h(x)$, tj. $L(\mathcal{M}') = h^{-1}(L(\mathcal{M}))$. □

2.3.2 Ekvivalence konečných automatů a regulárních gramatik

Na začátku této kapitoly již byla zmínka o tom, že pojem regulárního jazyka byl vlastně definován dvakrát – nejprve pomocí regulární gramatiky (viz část 1.2.2) a pak ještě jednou pomocí konečného automatu. V této části dokážeme, že tato nejednoznačnost je nezávadná, neboť třídy jazyků, které lze generovat regulárními gramatikami, resp. rozpoznat konečnými automaty, jsou stejné.

Začneme tím, že ukážeme, jak lze k dané regulární gramatice sestrojit ekvivalentní nedeterministický konečný automat (uvědomme si, že není podstatné, jestli jde o automat deterministický, nedeterministický, nebo dokonce s ε-kroky, protože všechny uvedené modely jsou navzájem ekvivalentní ve smyslu vět 2.43 a 2.48). Myšlenka důkazu je jednoduchá – stavy automatu budou odpovídat neterminálům gramatiky, tj. pro každý neterminál A bude existovat stav $\overline{A}$. Pro každé pravidlo $A \to aB$ přidáme do $\delta(\overline{A}, a)$ stav $\overline{B}$. Abychom se mohli vypořádat také s pravidly tvaru $C \to a$, zavedeme speciální koncový stav q_f, který přidáme do $\delta(\overline{C}, a)$. Počáteční stav bude $\overline{S}$, koncový stav q_f a případně také $\overline{S}$, pokud gramatika obsahuje pravidlo $S \to \varepsilon$.

Lemma 2.69. *Ke každé regulární gramatice $\mathcal{G} = (N, \Sigma, P, S)$ existuje nedeterministický konečný automat $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ takový, že $L(\mathcal{G}) = L(\mathcal{M})$.*

Důkaz. Položme

- $Q = \{\overline{A} \mid A \in N\} \cup \{q_f\}$, kde $q_f \notin N$.
- $q_0 = \overline{S}$.
- δ je nejmenší funkce $Q \times \Sigma \to 2^Q$ splňující:
  - Pokud $A \to aB$ je pravidlo v P, pak $\overline{B} \in \delta(\overline{A}, a)$.
  - Pokud $A \to a$ je pravidlo v P, kde $a \neq \varepsilon$, pak $q_f \in \delta(\overline{A}, a)$
- $F = \begin{cases} \{\overline{S}, q_f\} & \text{pokud } S \to \varepsilon \text{ je pravidlo v } P, \\ \{q_f\} & \text{jinak.} \end{cases}$

Nejprve dokážeme vztah mezi větnými formami $\mathcal{G}$ a rozšířenou přechodovou funkcí $\hat{\delta}$:

$$S \Rightarrow^* a_1 \dots a_k B, \text{ kde } a_1, \dots, a_k \in \Sigma, B \in N \iff \overline{B} \in \hat{\delta}(\overline{S}, a_1 \dots a_k), \text{ kde } \overline{B} \neq q_f$$

Důkaz provedeme indukcí vzhledem ke k:

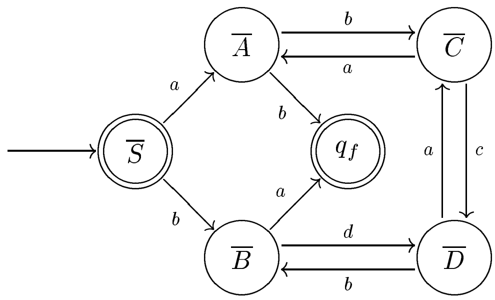
- **k = 0**: Pak nutně $B = S$, $\overline{B} = \overline{S}$ a obě strany dokazované ekvivalence (a tedy i ekvivalence samotná) platí.
- **Indukční krok:** Platí $S \Rightarrow^* a_1 \dots a_{k+1} B \iff S \Rightarrow^* a_1 \dots a_k C \Rightarrow a_1 \dots a_{k+1} B$ (tedy $C \to a_{k+1} B \in P$) $\iff \overline{C} \in \hat{\delta}(\overline{S}, a_1 \dots a_k)$ (indukční předpoklad), $\overline{B} \in \delta(\overline{C}, a_{k+1}) \iff \overline{B} \in \hat{\delta}(\overline{S}, a_1 \dots a_{k+1})$.

Nyní lze již snadno ukázat, že $w \in L(\mathcal{G}) \iff w \in L(\mathcal{M})$. Pokud $w = \varepsilon$, platí $\varepsilon \in L(\mathcal{G}) \iff S \to \varepsilon \in P \iff \overline{S} \in F \iff \varepsilon \in L(\mathcal{M})$. Nechť tedy $w \neq \varepsilon$, tj. $w = va$, kde $v \in \Sigma^*$, $a \in \Sigma$. Platí $va \in L(\mathcal{G}) \iff S \Rightarrow^* vB \Rightarrow va$ (tedy $B \to a \in P$) $\iff \overline{B} \in \hat{\delta}(\overline{S}, v), q_f \in \delta(\overline{B}, a) \iff q_f \in \hat{\delta}(\overline{S}, va) \iff va \in L(\mathcal{M})$. Je dobré si uvědomit, že poslední ekvivalence platí proto, že $\mathcal{M}$ nemůže akceptovat neprázdné slovo pomocí stavu $\overline{S}$ – pokud $\overline{S} \in F$, obsahuje P pravidlo $S \to \varepsilon$, a tedy S se nevyskytuje na pravé straně žádného pravidla; v automatu $\mathcal{M}$ pak do stavu $\overline{S}$ nevedou žádné přechody. □

Příklad 2.70. *Nechť $\mathcal{G} = (\{S, A, B, C, D\}, \{a, b, c, d\}, P, S)$, kde P obsahuje pravidla*

$$\begin{array}{ll} S \to aA \mid bB \mid \varepsilon & C \to aA \mid cD \\ A \to bC \mid b & D \to aC \mid bB \\ B \to dD \mid a & \end{array}$$

Ekvivalentní nedeterministický konečný automat má tento tvar:



Způsob, kterým lze naopak ke každému konečnému automatu sestrojit ekvivalentní regulární gramatiku, je do určité míry „inverzní" ke konstrukci použité v důkazu předchozího lemmatu. Neterminály budou odpovídat stavům, pravidla budou simulovat přechodovou funkci. Je tu však jeden problém – pokud automat přijímá prázdné slovo (tj. počáteční stav je koncovým stavem), musí každá ekvivalentní gramatika nutně obsahovat pravidlo $S \to \varepsilon$, kde S je kořen. Pak se ale S nesmí vyskytovat na pravé straně žádného pravidla (viz část 1.2.2). Přitom je ale možné, že některé přechody automatu končí v počátečním stavu a mají být simulovány pravidly, které mají na pravé straně S, což by vedlo ke konfliktu s požadavkem pravostranných výsktů S. Tento problém vyřešíme tak, že ke zkonstruované gramatice (mající jak $S \to \varepsilon$, tak i pravostranné výskyty S) lehce nalezneme ekvivaletní gramatiku splňující (jak vidno, v případě typu 3 čistě formální) požadavky definice z 1.2.2.

Lemma 2.71. *Pro každý konečný automat $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ existuje regulární gramatika $\mathcal{G} = (N, \Sigma, P, S)$ taková, že $L(\mathcal{M}) = L(\mathcal{G})$.*

Důkaz. Bez újmy na obecnosti předpokládejme, že $\mathcal{M}$ je nedeterministický. Nechť

- $N' = \{\overline{q} \mid q \in Q\}$.
- P' je nejmenší množina pravidel splňující:
  - Pokud $p \in \delta(q, a)$, je $\overline{q} \to a\overline{p}$ pravidlo v P'.
  - Pokud $p \in \delta(q, a)$ a $p \in F$, je $\overline{q} \to a$ pravidlo v P'.

Gramatika $\mathcal{G}' = (N', \Sigma, P', \overline{q_0})$ je zřejmě regulární, protože pravidla jsou předepsaného tvaru. Podobně jako v předchozím důkaze, tentokrát však jen pro neprázdná slova (problém s ε v případě $q_0 \in F$ vyřešíme posléze) ukažme platnost tvrzení:

$$\hat{\delta}(q_0, a_1 \dots a_k) \cap F \neq \emptyset, \text{ kde } k \geq 1, a_1, \dots, a_k \in \Sigma \iff \overline{q_0} \Rightarrow^* a_1 \dots a_k$$

Indukcí vzhledem k délce akceptovaného slova, tj. ke k:

- **k = 1**: pro $p \in F, p \in \hat{\delta}(q_0, a) \iff \overline{q_0} \to a$ je pravidlo v $P' \iff \overline{q_0} \Rightarrow^* a$.
- **Indukční krok:** pro $p \in F$, platí $p \in \hat{\delta}(q_0, a_1 \dots a_{k+1}) \iff$ existuje stav q takový, že $q \in \hat{\delta}(q_0, a_1 \dots a_k)$ a $p \in \delta(q, a_{k+1}) \iff \overline{q_0} \Rightarrow^* a_1 \dots a_k\overline{q}$ (podle indukčního předpokladu) a $\overline{q} \to a_{k+1}$ je pravidlo v $P' \iff \overline{q_0} \Rightarrow^* a_1 \dots a_{k+1}$.

Tedy pro každé $w \neq \varepsilon$ platí $w \in L(\mathcal{M}) \iff w \in L(\mathcal{G}')$. (*)

Je-li $q_0 \in F$, pak $\varepsilon \in L(\mathcal{M})$; v tomto případě $L(\mathcal{G}') = L(\mathcal{M}) \setminus \{\varepsilon\}$. Abychom obdrželi hledanou $\mathcal{G} = (N, \Sigma, P, S)$ položme

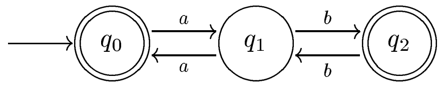
$N = N' \cup \{S\}$, kde $S \notin Q$ a

$P = P' \cup \{S \to \varepsilon\} \cup \{S \to \alpha \mid \overline{q_0} \to \alpha\}$.

Přidali jsme tedy nový neterminál (je nyní kořenem) a zajistili, aby jej bylo možné přepsat na cokoliv, na co se přepisoval kořen $\overline{q_0}$ (viz poslední sčítanec v definici P). Konečně do množiny pravidel jsme přidali kýžené $S \to \varepsilon$. Jelikož S je nově přidaný symbol, zcela jistě se nevyskytuje na žádné pravé straně v P. Zřejmě $L(\mathcal{G}) = L(\mathcal{G}') \cup \{\varepsilon\} = L(\mathcal{M})$.

Je-li $q_0 \notin F$, tj. $\varepsilon \notin L(\mathcal{M})$, pak stačí položit $N = N'$, $P = P'$ a $S = \overline{q_0}$. Požadované $L(\mathcal{G}) = L(\mathcal{M})$ je nyní jen jiným zápisem již dokázaného tvrzení (*). □

Příklad 2.72. *Mějme automat*



Použitím algoritmu z předchozího důkazu obdržíme ekvivalentní regulární gramatiku $\mathcal{G} = (\{S, \overline{q_0}, \overline{q_1}, \overline{q_2}\}, \{a, b\}, P, S)$, *kde P obsahuje pravidla*

$S \to a\overline{q_1} \mid \varepsilon$ $\qquad \overline{q_1} \to a\overline{q_0} \mid b\overline{q_2} \mid a \mid b$

$\overline{q_0} \to a\overline{q_1}$ $\qquad \overline{q_2} \to b\overline{q_1}$

Okamžitým důsledkem lemmat 2.69 a 2.71 je:

Věta 2.73. *Třídy jazyků, které lze generovat regulárními gramatikami, resp. rozpoznat konečnými automaty, jsou si rovny.*

2.3.3 Rozhodnutelné problémy pro třídu regulárních jazyků

Studujeme-li nějakou třídu jazyků (generovanou jistým typem gramatik či automatů), pak je zajímavé ptát se na existenci algoritmů (tzv. rozhodnutelnost či algoritmickou řešitelnost) jistých přirozených problémů. Máme-li dány dvě *libovolné* gramatiky (alternativně automaty) $\mathcal{G}$ a $\mathcal{G}'$ uvažovaného typu, pak nejtypičtějšími obvykle jsou tyto problémy:

1. ekvivalence: jsou $\mathcal{G}$ a $\mathcal{G}'$ ekvivaletní? Přesněji řečeno: existuje algoritmus, který pro dvě libovolné dané gramatiky $\mathcal{G}$ a $\mathcal{G}'$ (z uvažované třídy) rozhoduje, zda jsou ekvivaletní? Poznamejme, že obecně se nemusí jednat pouze o jazykovou ekvivalenci, tj. rovnost jazyků.
2. inkluse (jazyka): platí $L(\mathcal{G}) \subseteq L(\mathcal{G}')$? (opět ve výše uvedeném smyslu existence algoritmu – podobně i u všech dále uvedených problémů). Obecněji se jedná o problém tzv. simulace vzhledem k danému uspořádání.
3. příslušnost (slova k jazyku): je-li dáno libovolné slovo $w \in \Sigma^*$, platí $w \in L(\mathcal{G})$?
4. prázdnost (jazyka): je $L(\mathcal{G}) = \emptyset$?
5. universalita (jazyka): je $L(\mathcal{G}) = \Sigma^*$? (Σ je terminální abeceda $\mathcal{G}$)
6. konečnost (jazyka): je $L(\mathcal{G})$ konečný jazyk?
7. regularita (jazyka): je $L(\mathcal{G})$ regulární jazyk?[5] (či obecněji – srv. s poznámkou o ekvivaleci v 1: existuje ke $\mathcal{G}$ ekvivaletní regulární gramatika nebo konečný automat?)

V předchozím textu jsme se již setkali s celou řadou algoritmů, které sloužili převážně pro transformaci (tj. „překlad") mezi různými typy popisných formalismů. Samotný pojem „algoritmus" ovšem nebyl nijak blíže vysvětlen ani definován; spoléhali jsme (a spoléháme)

5. pro regulární jazyky je tento problém rozhodnutelný triviálně, což pro ostatní třídy neplatí

se na to, že je intuitivně jasný. Z ryze matematického hlediska je takový postup samozřejmě nepřípustný. Vše uvedeme na pravou míru v kapitole 5, kde je problematika formální definice algoritmu podrobněji rozebrána.

V této části budeme přitom předpokládat, že regulární jazyky jsou reprezentovány deterministickými konečnými automaty.

Věta 2.74. *Problém, zda libovolný daný regulární jazyk L nad abecedou Σ je prázdný, resp. roven Σ^*, je rozhodnutelný.*

Důkaz. Nechť $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ je deterministický konečný automat s totální přechodovou funkcí, rozpoznávající jazyk L. Zřejmě L je prázdný, právě když mezi dosažitelnými stavy automatu $\mathcal{M}$ není žádný prvek F; tuto podmínku lze algoritmicky ověřit, neboť množinu dosažitelných stavů $\mathcal{M}$ lze zkonstruovat užitím algoritmu 2.1.

Dále $L = \Sigma^*$, právě když co–$L = \emptyset$. Jak již víme, je třída regulárních jazyků uzavřena na komplement: jazyk co–L je rozpoznávaný deterministickým automatem $\overline{\mathcal{M}}$ – viz poznámka 2.11. Stačí tedy výše uvedeným způsobem otestovat prázdnost jazyka $L(\overline{\mathcal{M}})$. □

Poznámka 2.75. *Tvrzení předchozí věty lze též dokázat tak, že ukážeme platnost tvrzení:* Jazyk rozpoznávaný konečným automatem $\mathcal{M}$ o n stavech je neprázdný, právě když $\mathcal{M}$ akceptuje alespoň jedno slovo délky menší než n. *K důkazu lze využít přímo lemma o vkládání, resp. úvahy uvedené v jeho důkazu.*

Věta 2.76. *Problém, zda libovolný daný regulární jazyk L je konečný, resp. nekonečný, je rozhodnutelný.*

Důkaz. Nechť $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ je deterministický konečný automat, rozpoznávající jazyk L. Označme $n = card(Q)$. Dokážeme, že L je nekonečný, právě když $\mathcal{M}$ akceptuje alespoň jedno slovo $w \in \Sigma^*$ s vlastností $n \leq |w| < 2n$:

($\Longrightarrow$) Je-li L nekonečný, nutně obsahuje alespoň jedno slovo u délky alespoň n (de facto je takových slov samozřejmě nekonečně mnoho). Je-li $|u| \leq 2n$, jsme hotovi. V opačném případě lze slovo u rozdělit na tři části $u = xyz$ tak, že $1 \leq |y| \leq n$ a $xz \in L$ (viz lemma 2.13 o vkládání). Pokud je délka slova xz stále větší než $2n$, celý postup opakujeme; po konečném počtu opakování dostaneme slovo w požadovaných vlastností.

($\Longleftarrow$) Jelikož $|w| \geq n$, musí automat $\mathcal{M}$ při akceptování slova w projít dvakrát stejným stavem; slovo w lze tedy rozdělit na tři části $w = xyz$ tak, že $|y| \geq 1$ a platí $xy^iz \in L$ pro každé $i \in \mathbb{N}_0$ (viz důkaz lemmatu 2.13 o vkládání), tedy L je nekonečný.

To, zda $\mathcal{M}$ akceptuje alespoň jedno slovo w takové, že $n \leq |w| \leq 2n$, lze algoritmicky ověřit – těchto slov je konečně mnoho, můžeme tedy „vyzkoušet“ každé z nich. □

Věta 2.77. *Problém rovnosti libovolných daných regulárních jazyků je rozhodnutelný.*

Důkaz. Pro libovolné L_1, L_2 platí: $(L_1 = L_2) \iff (L_1 \cap \text{co–}L_2) \cup (\text{co–}L_1 \cap L_2) = \emptyset$. Pro L_1, L_2 regulární lze všechny uvedené operace algoritmicky realizovat (viz 2.11). Zbytek plyne z rozhodnutelnosti problému prázdnosti regulárního jazyka (2.74). □

Uvedený důkaz obsahuje i návod na konstrukci algoritmu pro rozhodování problému, zda $L_1 = L_2$. Nechť L_1, L_2 jsou po řadě rozpoznávány DFA $\mathcal{M}_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ a $\mathcal{M}_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ s totálními přechodovými funkcemi (je důležité, že oba automaty mají stejnou vstupní abecedu; tento předpoklad je bez újmy na obecnosti, neboť v opačném případě lze abecedy sjednotit a přechodové funkce znovu zúplnit). Položme $\mathcal{M}_\emptyset = (\mathcal{M}_1 \Cap \overline{\mathcal{M}_2}) \Cup (\overline{\mathcal{M}_1} \Cap \mathcal{M}_2)$. Pak $L_1 = L_2 \iff L(\mathcal{M}_\emptyset) = \emptyset$.

Uvažme však, že není nutné postupovat „otrocky“ po struktuře, jak je konstruován $\mathcal{M}_\emptyset$. Zřejmě $L(\mathcal{M}_\emptyset) \neq \emptyset \iff (L_1 \neq L_2) \iff \exists w \in (L_1 \cap \text{co–}L_2) \cup (\text{co–}L_1 \cap L_2)$. Hledejme tedy množinu R dosažitelných stavů synchronního paralelního spojení automatů $\mathcal{M}_1$ a $\mathcal{M}_2$ (viz poznámka 2.11). R bude obsahovat aspoň jednu dvojici (p, q) z množiny $F_1 \times (Q_2 - F_2) \cup (Q_1 - F_1) \times F_2$ právě když $\exists w \in (L_1 \cap \text{co–}L_2) \cup (\text{co–}L_1 \cap L_2) \iff (L_1 \neq L_2)$. Tedy definujme množinu $R \subseteq Q_1 \times Q_2$ takto:

$$R = \{(r, s) \mid \exists w \in \Sigma^* : r = \hat{\delta}_1(q_1, w) \ \wedge \ s = \hat{\delta}_2(q_2, w)\}$$

Právě jsme tedy ukázali, že $L(\mathcal{M}_1) \neq L(\mathcal{M}_2)$ právě když R obsahuje alespoň jednu dvojici stavů z $(F_1 \times (Q_2 - F_2)) \cup ((Q_1 - F_1) \times F_2)$, či ekvivaletně vyjádřeno:

Lemma 2.78. $L(\mathcal{M}_1) = L(\mathcal{M}_2)$ *právě když* R *obsahuje pouze takové dvojice, ve kterých jsou obě komponenty současně buď koncové nebo nekoncové stavy.*

Důkaz. Důkaz uvádíme jen pro snazší čtenářovu orientaci. Ukažme (srovnej s poznámkou 2.11 a s konstrukcí dosažitelných stavů v důkazu lemmatu 2.19), jak lze množinu R algoritmicky zkonstruovat; vyjdeme z toho, že R lze přirozeně aproximovat tak, že v její definici položíme omezení na délku slova w. Obdržíme tak systém množin R_i, kde $i \in \mathbb{N}_0$, definovaný takto:

$$R_i = \{(r, s) \mid \exists w \in \Sigma^* : |w| \leq i \ \wedge \ r = \hat{\delta}_1(q_1, w) \ \wedge \ s = \hat{\delta}_2(q_2, w)\}$$

Platí tedy

$$R = \bigcup_{i=0}^{\infty} R_i \tag{2.7}$$

Každou z množin R_i lze ovšem snadno zkonstruovat na základě induktivního předpisu:

- $R_0 = \{(q_1, q_2)\}$
- $R_{i+1} = R_i \cup \{(\delta_1(r, a), \delta_2(s, a)) \mid (r, s) \in R_i \ \wedge \ a \in \Sigma\}$

Označme $n = \mathit{card}(Q_1 \times Q_2)$. Jelikož $R_i \subseteq R_{i+1}$ a každá z množin R_i je podmnožinou $Q_1 \times Q_2$, nutně existuje $k \leq n$ takové, že $R_k = R_{k+1}$. Z induktivního předpisu pro R_i je vidět, že množina R_{k+1} závisí pouze na množině R_k – proto dokonce platí $R_k = R_{k+j}$ pro libovolné $j \in \mathbb{N}_0$. Vztah 2.7 lze tedy přepsat do tvaru

$$R = \bigcup_{i=0}^{k} R_i = R_k \tag{2.8}$$

Tím je formálně dokázána správnost i konečnost algoritmu 2.4. □

Algoritmus 2.4 Test rovnosti regulárních jazyků.

Vstup: DFA $\mathcal{M}_j = (Q_j, \Sigma, \delta_j, q_j, F_j), j = 1, 2$ s totálními δ_j.
Výstup: **YES** jestliže $L(\mathcal{M}_1) = L(\mathcal{M}_2)$, **NO** jinak.

$i := 0;\ \ R_0 := \{(q_1, q_2)\};$
repeat
  $R_{i+1} := R_i \cup \{(\delta_1(r, a), \delta_2(s, a)) \mid (r, s) \in R_i \ \wedge\ a \in \Sigma\};$
  if R_{i+1} obsahuje dvojici $(p, q),$ kde $p \in F_1 \not\Longleftrightarrow q \in F_2$ **then**
    return NO;
  end if
  $i := i + 1;$
until $R_i = R_{i-1}$
return YES;

Algoritmus 2.4 je ve skutečnosti mírně zoptimalizovaný; test, zda R obsahuje „nedovolené“ dvojice, se provádí po každé iteraci a pokud uspěje, algoritmus se ihned ukončí.

Alternativně lze větu 2.77 (i s implicitně obsaženým algoritmem) dokázat takto: nechť L_1 a L_2 jsou regulární jazyky, po řadě rozpoznávané DFA $\mathcal{M}_1$ a $\mathcal{M}_2$. Ke každému z nich lze *sestrojit* minimální DFA $\mathcal{M}'_1$, resp. $\mathcal{M}'_2$. Pak $L_1 = L_2$ právě když $\mathcal{M}'_1$ a $\mathcal{M}'_2$ jsou isomorfní (liší se jen pojmenováním stavů). Ověření tohoto isomorfismu, jako isomorfismu dvou přechodových grafů s konečně mnoha stavy (uzly), je jistě algoritmicky realizovatelné, či jinak řečeno rozhodnutelné. Tím je důkaz ukončen.

Abychom však výše zmíněný isomorfismus nemuseli ověřovat zkoumáním všech možností, je vhodné si uvědomit, že každý automat (a tedy i $\mathcal{M}'_1$ a $\mathcal{M}'_2$ z výše uvedeného důkazu) lze převést do jistého standardizovaného (tzv. *kanonického*) tvaru. Můžeme totálně uspořádat vstupní abecedu $\Sigma = \{a_1, \ldots a_m\}, a_i < a_{i+1}$ a stavy postupně systematicky očíslovat: počátečnímu stavu přiřadíme číslo 1 a postupně procházíme všechny jeho následníky, tj. nejprve pro a_1 atd. až po a_m. Pokud najdeme následníka, který ještě nemá přiřazeno žádné číslo, přiřadíme mu nové, dosud nepoužité číslo; například bylo-li poslední přiřazené číslo i, přiřadíme jako nové číslo $i + 1$. Pokud najdeme následníka, který již přiřazeno číslo má, neděláme nic. Takto zpracujeme všechny stavy, a to postupně dle jim přiřazených čísel. Detaily tohoto algoritmu ponecháváme čtenáři. Jen si uvědomme, že časově náročný test na isomorfismus dvou přechodových grafů lze převodem do kanonického tvaru nahradit jednoduchým testem na identitu přechodových grafů odpovídajících automatů v kanonickém tvaru.

2.4 Aplikace regulárních jazyků a konečných automatů

S omluvou: jen stručně (výčtem) zmiňme několik z mnoha aplikačních oblastí.

Vyhledávání vzorů (tzv. pattern matching) – například vyhledávání v textu (editory, textové systémy), v DNA sekvencích a v celé řadě dalších oblastí. Například v Unixu se jedná o programy

`grep` - vyhledávání podle zadaného regulár. výrazu (RE); algoritmus implementující NFAů

`egrep` - vyhledávání podle zadaného tzv. rozšířeného RE (viz uzavřenost na průnik a komplement); rychlý algoritmus(DFA), který může mít exponenciálně mnoho stavů.

`fgrep` - vyhledávání podle zadaného řetězce (rychlý, paměťově nenáročný DFA, jehož základem je tzv. Knuth-Morris-Prattův algoritmus – jedna z perel mezi algoritmy).

Další použití regulárních výrazů je například v emacs, vi, sed, sh, ...

Rozpoznávání a zpracování lexikálních jednotek, například při automatizované konstrukci lexikálních analyzátorů v překladačích programovacích jazyků, v počítačové lingvistice a dalších. Opět v OS Unix jde o program `lex` (či jeho uživatelsky komfortnější variant `flex`) - generuje C programy (deterministické FA), které mají být použity při nejrůznějším zpracování lexikálních jednotek.

Specifikace a verifikace konečně stavových systémů, jak hardwerových, tak i softwarových, například komunikačních a řídicích protokolů.

Zpracování obrazů (image processing). Jednoduché, ale velmi silné rozšíření FA na tzv. vážené konečné automaty (weighted FA), kdy přechodům a stavům jsou přiřazeny jistá čísla – váhy (např. u stavu určuje stupeň šedi daného pixelu).

Konečné automaty nad nekonečnými slovy - reaktivní (paralelní a distribuované) systémy.

Konečné automaty s výstupem, tzv. *překladové* automaty - návrh hardwarových obvodů a řada dalších.

Kapitola 3

Bezkontextové jazyky a zásobníkové automaty

V této kapitole se budeme podrobněji zabývat bezkontextovými gramatikami (dále často jen CFGs či CFG pro singulár) a jazyky jimi generovanými – bezkontextovými jazyky (CFL). Podobně jako regulární jazyky mají též CFL i značný praktický význam, například při definici syntaxe programovacích jazyků, formalizaci pojmu syntaktická analýza a návrhu překladu programovacích jazyků a dalších. Pro ilustraci uveďme, že pomocí CFGs jsme schopni popsat dobře uzávorkované aritmetické výrazy, blokovou strukturu v programovacích jazycích či obdobnou strukturu například v sázecím systému LaTeX(tj. dobré uzávorkování pomocí závorek **begin** a **end**, ..., závorky typů \begin{itemize}, \end{itemize}, ..., \begin{description}, \end{description} – obecně tedy půjde o popis dobrého uzávorkování jazyka obsahujího závorky $(_1,)_1, \ldots, (_n,)_n, n \geq 1$). Žádný z těchto rysů nelze popsat pomocí regulárních gramatik, jak již ostatně víme z předchozí kapitoly (jazyk $\{0^n 1^n | n \geq 0\}$ není regulární).

V části 3.2 zavedeme tzv. zásobníkové automaty, které akceptují právě CFL a ukážeme základní principy, na nichž jsou založeny transformace CFG na (jazykově) ekvivaletní zásobníkové automaty a naopak. Na závěr této kapitoly pak budeme v 3.3 studovat některé základní výsledky široce rozpracované teorie CFL, které nám například umožní vhodně upravovat/transformovat CFG, rozhodovat, zda daný jazyk je či zejména není CFL a další užitečné vlastnosti této třídy jazyků.

3.1 Bezkontextové gramatiky

3.1.1 Derivační stromy

V gramatice je možné, aby různé derivace byly ekvivaletní v tom smyslu, že všechny tyto derivace používají stejná pravidla na stejných místech, tj. jsou aplikována na stejné výskyty neterminálů, avšak v různém pořadí. Zatímco definice takové ekvivalence je pro gramatiky typu 0 poněkud komplikovaná, lze v případě bezkontextových gramatik zavést pro třídu ekvivalentních derivací jednoduchou grafovou reprezentaci, tzv. derivační strom (někdy též zvanou strom odvození). Alternativně lze reprezentovat ve výše uvedeném smyslu ekvivalentní derivace jistými *kanonickými* derivacemi, v nichž aplikujeme pravidla jistým standardizovaným postupem (např. v každém kroku derivace přepisujeme ve větné formě nejlevější neterminál, resp. nejpravější neterminál. Každou takovou derivaci nazveme *levou* resp. *pravou derivací*.

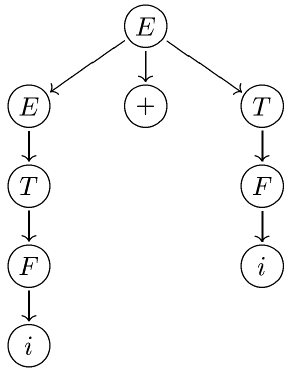
Uzly derivačního stromu v libovolné dané CFG $\mathcal{G}$ jsou označeny návěštími, která jsou buď terminály nebo neterminály, případně ε. Má-li vnitřní uzel n návěští A a jeho bezprostřední následníci (dále jen synové) mají zleva doprava po řadě návěští $X_1, \ldots, X_n$, pak $A \to X_1, \ldots, X_n$ musí být pravidlem v $\mathcal{G}$. Formálně řečeno:

Definice 3.1. Nechť $\mathcal{G} = (N, \Sigma, P, S)$ je CFG. Strom T nazveme *derivačním stromem* v $\mathcal{G}$ právě když platí tyto podmínky:

1. každý uzel má návěští, které je symbolem z $N \cup \Sigma \cup \{\varepsilon\}$,
2. kořen má návěští S,
3. má-li vnitřní uzel návěští A, pak $A \in N$,
4. má-li uzel n návěští A a jeho všichni synové $n_1, \ldots, n_k$ mají v uspořádání zleva doprava návěští $X_1, \ldots, X_k$, pak $A \to X_1 \ldots X_k \in P$,
5. má-li uzel n návěští ε, pak n je list a je jediným synem svého otce.

Výsledkem derivačního stromu T nazveme slovo vzniklé zřetězením návěští listů v uspořádání zleva doprava.

Příklad 3.2. *Nechť $\mathcal{G}_0$ je gramatika s pravidly*
$$\begin{array}{lcll} E & \to & E+T & \mid \; T \\ T & \to & T*F & \mid \; F \\ F & \to & (E) & \mid \; i \end{array}$$
pak derivační strom



reprezentuje deset vzájemně ekvivalentních derivací, například

1. $E \Rightarrow E+T \Rightarrow T+T \Rightarrow F+T \Rightarrow i+T \Rightarrow i+F \Rightarrow i+i$ *nebo*
2. $E \Rightarrow E+T \Rightarrow E+F \Rightarrow E+i \Rightarrow T+i \Rightarrow F+i \Rightarrow i+i$ *a též*
3. $E \Rightarrow E+T \Rightarrow T+T \Rightarrow T+F \Rightarrow F+F \Rightarrow F+i \Rightarrow i+i$ *a další.*

Všimněme si, že 1 je levá, kdežto 2 je pravá derivace.

Věta 3.3. *Nechť $\mathcal{G} = (N, \Sigma, P, S)$ je CFG. Pak pro libovolné $\alpha \in (N \cup \Sigma)^*$ platí $S \Rightarrow^* \alpha$ právě když v $\mathcal{G}$ existuje derivační strom s výsledkem α.*

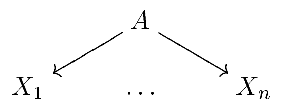
Důkaz. Je-li $\mathcal{G} = (N, \Sigma, P, S)$ CFG a $A \in N$, definujmeme $\mathcal{G}_A \stackrel{def}{=} (N, \Sigma, P, A)$, tj. gramatiku s týmiž pravidly jako $\mathcal{G}$, která však má kořen A. Důkaz povedeme tak, že nejprve ukážeme silnější tvrzení:

$$\forall A \in N.\ (A \Rightarrow^* \alpha \Longleftrightarrow \text{v } \mathcal{G}_A \text{ existuje derivační strom s výsledkem } \alpha). \qquad (*)$$

Tvrzení věty pak obdržíme specializací $A = S$.

I. ($\Longleftarrow$) Předpokládejme, že α je výsledkem derivačního stromu, který má k vnitřních uzlů a indukcí vzhledem ke k ukažme, že pak $A \Rightarrow^* \alpha$.

1. $k = 1$. Existuje-li ve stromu jediný vnitřní uzel,



pak pro $\alpha = X_1 \dots X_n$ z definice derivačního stromu (bod 3. a 4.) plyne, že $A \to \alpha \in P$.

2. $k > 1$. Předpokládejme (IP), že dokazované tvrzení platí pro stromy, které mají nanejvýš $k - 1$ vnitřních uzlů.

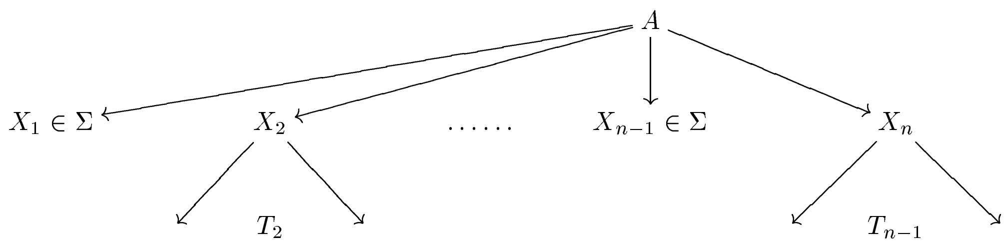
Nechť α je výsledkem stromu s k vnitřními uzly. Následníci kořene (označme je $1, \dots n$) nejsou jen samé listy (mimo kořen zde musí být ještě aspoň jeden vnitřní uzel, protože $k > 1$) a nechť jejich návěští jsou v uspořádání zleva po řadě $X_1 \dots X_n$. Pak jistě $A \to X_1 \dots X_n \in P$. Nyní:

(a) Je-li uzel i vnitřním uzlem, pak je současně kořenem nějakého podstromu T_i a $X_i \in N$. Podstrom T_i má nejvýše $k - 1$ vnitřních uzlů a je stromem v $\mathcal{G}_{X_i}$ s výsledkem α_i. Dle IP je $X_i \Rightarrow^* \alpha_i$.

(b) Je-li uzel i listem, pak $X_i = \alpha_i$ (tedy triviálně $X_i \Rightarrow^* \alpha_i$).

Zřejmě $\alpha = \alpha_1 \dots \alpha_n$, a tedy celkem dostáváme

$$\begin{array}{lll} A \Rightarrow & X_1 X_2 \dots X_n & \\ \Rightarrow^* & \alpha_1 X_2 \dots X_n & (\text{ dle IP, je-li } X_1 \in N;\ \text{triv. pro } X_1 \in \Sigma) \\ & \vdots & \\ \Rightarrow^* & \alpha_1 \alpha_2 \dots \alpha_{n-1} X_n & (\text{ dtto }) \\ \Rightarrow^* & \alpha_1 \alpha_2 \dots \alpha_n \ = \alpha & \text{tj.} A \Rightarrow^* \alpha \end{array}$$



II. ($\Longrightarrow$) Předpokládejme, že $A \Rightarrow^* \alpha$ a máme ukázat,že v $\mathcal{G}_A$ existuje derivační strom s výsledkem α. Tentokrát použijeme indukci vzhledem k délce odvození $A \Rightarrow^* \alpha$.

1. Je-li $A \Rightarrow \alpha$, tj. v jednom kroku, pak $A \to \alpha \in P$ a z definice derivačního stromu dostáváme existenci stromu s výsledkem α.

2. Předpokládejme (IP), že je-li $A \Rightarrow^* \alpha$ v méně než k krocích, pak v $\mathcal{G}_A$ existuje derivační strom s výsledkem α (IP). Nechť tedy $A \Rightarrow^* \alpha$ v k krocích a nechť 1. krok je tvaru $A \to X_1 \dots X_n$. Zřejmě každý symbol v α je buď některé z $X_1 \dots X_n$ nebo je symbolem v řetězu odvoditelného z některého z nich, a to v méně než k krocích (dle IP platí pro něj dokazované tvrzení). Dále, ta část α, která je odvoditelná z X_i leží vlevo od té části α, která je odvoditelná z X_j pro $i < j$. Můžeme tedy psát $\alpha = \alpha_1 \dots \alpha_n$, kde $X_i \Rightarrow^* \alpha_i$ a označme takový podstrom T_i.

Hledaný derivační strom nyní zkonstruujeme takto: začneme s konstrukcí odpovídající prvnímu kroku odvození, tj. $A \Rightarrow X_1 \dots X_n$ a dostaneme

$$\begin{tikzcd}
 & A \arrow[dl] \arrow[dr] & \\
X_1 & \dots & X_n
\end{tikzcd}$$

a dále každé X_i nahradíme stromem T_i (je-li X_i terminál, je náhrada triviální – nic nenahrazujeme). Výsledkem takto vzniklého stromu je zřejmě α.

Tím jsme dokázali tvrzení (*); tvzení věty obdržíme (specializací) tak, že v (*) položíme $A = S$ ($\mathcal{G}_S = \mathcal{G}$). □

Specielně tedy pro terminální řetěz w platí, že $w \in L(\mathcal{G})$ právě když v $\mathcal{G}$ existuje derivační strom s výsledkem w.

Není těžké ukázat, že každému derivačnímu stromu v CFG odpovídá jediná levá derivace a obráceně, každé levé derivaci odpovídá jediný derivační strom. (Například v důkazu předchozí věty byla k derivačnímu stromu s kořenem A a výsledkem α nalezena levá derivace větné formy α z A za předpokladu, že každá z $X_i \Rightarrow^* \alpha_i$ byla levou derivací.) Analogické tvrzení platí o vzájemně jednoznačné korespondenci mezi derivačními stromy a pravými derivacemi (a tedy i mezi levými a pravými derivacemi).

Každý, kdo programuje v jazyce Pascal, jistě ví, že existují CF gramatiky, v nichž má jedna věta či větná forma několik různých derivačních stromů.

Příklad 3.4. *Mějme gramatiku s pravidly*

$$S \to \textbf{\textit{if}}\ b\ \textbf{\textit{then}}\ S\ \textbf{\textit{else}}\ S \mid \textbf{\textit{if}}\ b\ \textbf{\textit{then}}\ S \mid a$$

Pak například věta ***if*** b ***then if*** b ***then*** a ***else*** a *má dva různé derivační stromy, které odpovídají interpretaci* ***if*** b ***then*** *(****if*** b ***then*** a*)* ***else*** a *resp.* ***if*** b ***then*** *(****if*** b ***then*** a ***else*** a*). Zkonstruujte oba stromy!*

Definice 3.5. CFG $\mathcal{G}$ se nazývá *víceznačná (nejednoznačná)* právě když existuje $w \in L(\mathcal{G})$ mající alespoň dva různé derivační stromy. V opačném případě říkáme, že $\mathcal{G}$ je jednoznačná. Jazyk L se nazývá *vnitřně (inherentně) víceznačný* právě když každá gramatika, která jej generuje, je víceznačná.

Příklad 3.6. *Gramatika* $\mathcal{G}_1$ *s pravidly* $E \to E + E \mid E * E \mid (E) \mid i$ *, která je ekvivaltní s gramatikou* $\mathcal{G}_0$ *z příkladu 3.2, je víceznačná; například proto, že věta* $i + i + i$ *má dvě různé levé derivace a jim odpovídající dva různé derivační stromy:*

$$\begin{tikzcd}
 & & E \arrow[dl] \arrow[d] \arrow[dr] & \\
 & E \arrow[dl] \arrow[d] \arrow[dr] & + & E \arrow[d] \\
E \arrow[d] & + & E \arrow[d] & i \\
i & & i &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & E \arrow[dl] \arrow[d] \arrow[dr] & & \\
E \arrow[d] & + & E \arrow[dl] \arrow[d] \arrow[dr] & \\
i & E \arrow[d] & + & E \arrow[d] \\
 & i & & i
\end{tikzcd}$$

Nejednoznačnost gramatiky může v některých praktických aplikacích působit jisté obtíže: pokud nalezení derivačního stromu věty (například zdrojového textu programu) je základem pro stanovení významu věty, vznikl by v případě nejednoznačné gramatiky problém, který z těchto významů zvolit (nehledě k nárůstu časové i paměťové složitosti spojené s hledáním všech derivačních stromů).

Ke gramatice $\mathcal{G}_1$ z příkladu 3.6 lze zkonstruovat ekvivaletní jednoznačnou gramatiku $\mathcal{G}_2$ s pravidly například $E \to E{+}T \mid E{*}T \mid T\,;\; T \to (E) \mid i$. Všimněme si však, že $\mathcal{G}_0$ z příkladu 3.2, na rozdíl od $\mathcal{G}_2$, umožňuje postihnout (již na syntaktické úrovni) asociativitu tak, aby reflektovala i obvyklou prioritu operátorů, tj. že $*$ váže silněji než $+$; věta $i + i * i$ by v $\mathcal{G}_2$ byla asociována zleva jako $(i + i) * i$.

Poznamejme, že ne vždy lze k dané gramatice (resp. jazyku) nalézt ekvivaletní gramatiku, která by byla jednoznačná. Takovým vnitřně víceznačným jazykem je například $L = \{a^i b^j c^k \mid i = j \text{ nebo } j = k\}$. Intiutivně řečeno, každá gramatika generující L musí být schopna vytvářet jistou sadou pravidel jak slova spňující $i = j$, tak i jinou sadou pravidel slova s $j = k$, a tudíž nelze zabránit tomu, aby aspoň některá ze slov $a^i b^i c^i$, tj. $i = j = k$ nebyla generovatelná oběma různými způsoby. Jak ukážeme později, bohužel ani pro problém, zda libovolná daná CFG je (ne)jednoznačná neexistuje algoritmus.

3.1.2 Transformace bezkontextových gramatik

Jak jsme již naznačili v závěru minulé sekce, bývá často výhodné modifikovat danou gramatiku $\mathcal{G}$ tak, aby vytvářela takovou stukturu vět z $L(\mathcal{G})$, která by zajistila splnění některých kýžených vlastností jazyka či gramatiky. Mimo již zmíněné asociativity a jednoznačnosti je těchto vlastností (a jim korespondujících transformací) gramatik celá řada.

Definice 3.7. Řekneme, že symbol $X \in N \cup \Sigma$ je *nepoužitelný* v CFG $\mathcal{G} = (N, \Sigma, P, S)$ právě když v $\mathcal{G}$ neexistuje derivace tvaru $S \Rightarrow^* wXy \Rightarrow^* wxy$ pro nějaká $w, x, y \in \Sigma^*$. Řekneme, že $\mathcal{G}$ je *redukovaná*, jestliže neobsahuje žádné nepoužitelné symboly.

Poznámka 3.8. *Povšimněme si, že výše uvedená definice postihuje nepoužitelnost dvojího druhu: neexistence první části uvedené derivace říká, že symbol $X \in N \cup \Sigma$ se nevyskuje v žádné větné formě (jedná se o tvz. nedosažitelný symbol), kdežto neexistence druhé části zmíněné derivace vyjadřuje skutečnost, že z neterminálu X nelze vyderivovat žádný terminální řetěz (tzv. nenormovaný, někdy též redundatní neterminál [1]). Poznamejme ještě, že existence obou zmíněných derivací ještě není postačující podmínkou pro použitelnost symbolu: X se může totiž objevit jen v takové větné formě, která obsahuje nenormovaný neterminál.*

Pokud z $\mathcal{G}$ nepoužitelné symboly vypustíme, pak se $L(\mathcal{G})$ zřejmě nezmění. Řešme nejprve druhý z těchto problémů, tj. zda $\{w \in \Sigma^* \mid A \Rightarrow^* w\} = \emptyset$. Pokud pro tento problém nalezneme algoritmus, pak jeho existence implikuje (položením $A = S$) existenci algoritmu pro problém zda $L(\mathcal{G}) = \emptyset$, či nikoli.

Věta 3.9. *Algoritmus 3.1 "Je $L(\mathcal{G})$ neprázdný?" vrací "ANO"* $\iff \exists w \in \Sigma^*.\ S \Rightarrow^* w$.

---

1. Normou neterminálu A obvykle rozumíme délku nejkratšího terminálního řetězu odvoditelného z A, pokud taková derivace existuje

Algoritmus 3.1 Je $L(\mathcal{G})$ neprázdný?

Vstup: CFG $\mathcal{G} = (N, \Sigma, P, S)$.
Výstup: "ANO" je-li $L(\mathcal{G}) \neq \emptyset$; "NE" v ostatních případech

Dle následujících kroků konstruuj induktivně množiny $N_0, N_1, \ldots$ takto:
$i := 0;\ N_0 := \emptyset;$ (* inicializace *) (1)
repeat
$i := i + 1;$ (* iterace *) (2)
$N_i := N_{i-1} \cup \{A | A \to \alpha \in P, \alpha \in (N_{i-1} \cup \Sigma)^*\}$ (3)
until $N_i = N_{i-1};$ (* test ukončení *) (4)
$N_e := N_i;$ (5)
if $S \in N_e$ **then** output("ANO") **else** output("NE"). (6)

Důkaz. Poznamejme, že každá N_i je definována jako množina neterminálů, které lze v nejvýše i krocích přepsat na řetěz terminální. Dokažme nejprve (opět) obecnější tvrzení:

$$A \in N_e \iff \exists w \in \Sigma^*. A \Rightarrow^* w. \qquad (*)$$

I.($\Longrightarrow$) $A \in N_e \Longrightarrow \exists i.\ A \in N_i$. Indukcí dokažme tvrzení:

$$\exists i.\ A \in N_i \Longrightarrow \exists w \in \Sigma^*.\ A \Rightarrow^* w$$

1. $i = 0$: platí triviálně, protože $N_0 = \emptyset$. (viz řádek (1) v Algoritmu 3.1)
2. $i > 0$ (IP): Předpokládejme, že dokazované tvrzení platí pro i.
Nechť nyní $A \in N_{i+1}$:

- je-li A rovněž prvkem z N_i, pak tvrzení plyne přímo z (IP).
- je-li $A \in N_{i+1} \setminus N_i$, pak existuje $A \to X_1 \ldots X_k \in P$, kde každé $X_j (1 \leq j \leq k)$ je buď terminál, nebo neterminál patřící do N_i (viz řádek (3)). Tedy existují w_j tak, že $X_j \Rightarrow^* w_j$ pro všechna $j \in \langle 1, k \rangle, w_j \in \Sigma^*$ (je-li $X_j \in \Sigma$, pak $w_j = X_j$, jinak existence w_j plyne z (IP)).

Tedy celkem $A \Rightarrow X_1 \ldots X_k \Rightarrow^* w_1 X_2 \ldots X_k \Rightarrow^* \ldots \Rightarrow^* w_1 \ldots w_k,\ w_1 \ldots w_k \in \Sigma^*$

II. ($\Longleftarrow$) Definice množin N_i zajišťuje, že pokud nastane $N_i = N_{i-1}$, pak platí $N_i = N_{i+1} = \cdots$. Máme ukázat, že pokud $A \Rightarrow^* w$ pro nějaké $w \in \Sigma^*$, pak $A \in N_e$, přičemž na základě předchozí poznámky stačí ukázat, že $A \in N_i$ pro nějaké i. Tedy indukcí dokažme:

$$A \overset{n}{\Rightarrow} w, w \in \Sigma^* \Longrightarrow A \in N_i \text{ pro nějaké } i.$$

1. $n = 1$, tj. $A \to w \in P, w \in \Sigma^*$ okamžitě dává $i = 1$ (viz řádek (3) v Algoritmu 3.1).
2. $n > 1$ (IP): Předpokládejme, že dokazované tvrzení platí pro všechna n a nechť nyní $A \overset{n+1}{\Rightarrow} w$. Pak zřejmě tuto derivaci lze rozepsat do tvaru $A \Rightarrow X_1 \ldots X_k \overset{n}{\Rightarrow} w$, kde $w = w_1 \ldots w_k$ takové, že $X_j \overset{n_j}{\Rightarrow} w_j$ pro všechna j a kde $n_j \leq n$.

Je-li $X_j \in N$, pak dle (IP) je $X_j \in N_{i_j}$ pro nějaké i_j. Je-li $X_j \in \Sigma$, pak $i_j = 0$. Položme $i = 1 + \max\{i_1, \ldots, i_k\}$. Pak zřejmě $A \in N_i$, čímž je důkaz indukcí ukončen.

Položíme-li $A = S$ v právě dokázaném tvrzení (*), dostáváme tvrzení věty. $\square$

Poznamejme, že jsme právě dokázali, že pokud algoritmus 3.1 zastaví, pak dává koretkní odpověd. Důkaz, že algoritmus musí skončit, a to nejpozději po $n + 1$ iteracích (pro $n = \mathrm{card}(N)$), plyne okamžitě z faktu, že $N_e \subseteq N$ – viz monotonie N_i vzhledem k inklusi (během iterace platí $N_{i-1} \subseteq N_i$) a tvaru testu ukončení. Celkem tedy máme:

Důsledek 3.10. *Existuje algoritmus, který pro libovolnou danou CFG $\mathcal{G}$ rozhoduje, zda $L(\mathcal{G}) = \emptyset$.*

Pro eliminaci nepoužitelných symbolů musíme ještě umět odstranit nedosažitelné symboly (viz Poznámka 3.8). Tuto činnost provádí Algoritmus 3.2.

Algoritmus 3.2 Eliminace nedosažitelných symbolů

Vstup: CFG $\mathcal{G} = (N, \Sigma, P, S)$.
Výstup: CFG $\mathcal{G}' = (N', \Sigma', P', S)$ bez nedosažitelných symbolů: $L(\mathcal{G}) = L(\mathcal{G}')$

Dle následujících kroků konstruuj induktivně množiny $V_0, V_1, \dots$ takto:
$i := 0;\ V_i := \{S\};$ (* inicializace *) (1)
repeat
$i := i + 1;\ \ V_i := V_{i-1} \cup \{X \mid \exists A.(A \in V_{i-1} \land A \to \alpha X \beta \in P)\}$ (* iterace *) (2)
until $V_i = V_{i-1}$; (* test ukončení *) (3)
$N' := N \cap V_i;\ \Sigma' := \Sigma \cap V_i;\ P' := P \cap (V_i \times V_i^*)$ (4)

Ke korektnosti algoritmu 3.2 zbývá uvážit, že jeho ukončení je implikováno faktem, že $V_i \subseteq N \cup \Sigma$, a tedy iteraci (2) – (3) lze provést jen konečně mnohokrát. Důkaz, že při skončení $\mathcal{G}'$ neobsahuje nedosažitelné symboly spočívá v ukázání, že $S \Rightarrow^* \alpha X \beta \iff \exists i.\ X \in V_i$ (indukcí vzhledek k i). Formální důkaz je ponechán čtenáři jako cvičení.

Nyní jsme v situaci, kdy můžeme prezentovat algoritmus 3.3, který odstraňuje z CFG nepoužitelné symboly.

Algoritmus 3.3 Eliminace nepoužitelných symbolů

Vstup: CFG $\mathcal{G} = (N, \Sigma, P, S)$ taková, že $L(\mathcal{G}) \neq \emptyset$.
Výstup: CFG $\mathcal{G}' = (N', \Sigma', P', S)$ bez nepoužitelných symbolů: $L(\mathcal{G}) = L(\mathcal{G}')$

Použij algoritmus 3.1 se vstupem $\mathcal{G}$ a s výstupem N_e; (1)
Polož $\mathcal{G}_1 = (N \cap N_e, \Sigma, P_1, S)$, kde $P_1 := P \cap (N_e \times (N_e \cup \Sigma)^*)$;
Použij algoritmus 3.2 se vstupem $\mathcal{G}_1$; výstupem je $\mathcal{G}' = (N', \Sigma', P', S)$ (2)

Krok (1) algoritmu 3.3 odstraňuje z $\mathcal{G}$ všechny neterminály, které nemohou vygenerovat terminální řetěz, krok (2) odstraní nedosažitelné symboly, tj. každý $X \in N' \cup \Sigma'$ se musí vyskytnout alespoň jednou v nějaké derivaci tvaru $S \Rightarrow^* wXy \Rightarrow^* wxy$. Konečně poznamenejme, že záměna pořadí kroků (1) a (2) obecně nevede k cíli (proč?).

Věta 3.11. *Každý neprázdný CFL je generován nějakou redukovanou CFG (tj. CFG bez nepoužitelných symbolů).*

Důkaz. Nechť $L = L(\mathcal{G})$ je neprázdný CFL a nechť $\mathcal{G}_1$ a $\mathcal{G}'$ jsou, jak uvedeno v algoritmu 3.3. Zřejmě $L(\mathcal{G}) = L(\mathcal{G}')$ (kompozice transformací zachovávajích ekvivalenci).

Předpokládejme, že $\mathcal{G}'$ má nepoužitelný symbol X. Pak ovšem v $\mathcal{G}'$ existuje derivace $S \Rightarrow^* \alpha X \beta$ (viz krok (2)). Jelikož všechny symboly z $\mathcal{G}'$ jsou též v $\mathcal{G}_1$, pak (viz krok (1)) pro nějaký terminální řetěz platí $S \Rightarrow^* \alpha X \beta \Rightarrow^* w$, a tedy žádný symbol z derivace $\alpha X \beta \Rightarrow^* w$ není krokem (2) eliminován. Tedy z X lze v $\mathcal{G}'$ odvodit terminální řetěz, což vede ke sporu s předpokladem, že X je nepoužitelný. □

Příklad 3.12. *Nechť $\mathcal{G}$ má pravidla $\{S \to a|A, A \to AB, B \to b\}$ a aplikujme na ni algoritmus 3.3. Po kroku (1) máme $N_e = \{S, B\}$, takže $\mathcal{G}_1 = (\{S, B\}, \{a, b\}, \{S \to a, B \to b\}, S)$; po kroku (2) obdržíme $V_2 = V_1 = \{S, a\}$, a tedy $\mathcal{G}' = (\{S\}, \{a\}, \{S \to a\}, S\}$. Poznamenejme, že pokud bychom na $\mathcal{G}$ použili nejprve krok (2), tj. algoritmus 3.2, shledali bychom, že všechny symboly jsou dosažitelné – $\mathcal{G}$ by se vůbec nezměnila. Následná aplikace kroku (1) by dala $N_e = \{S, B\}$, a tudíž bychom celkem dostali $\mathcal{G}_1$ a nikoli $\mathcal{G}'$.*

Nyní se budeme zabývat eliminací pravidel tvaru $A \to \varepsilon$, tzv. ε-pravidly. Pokud však $L(\mathcal{G})$ obsahuje ε, pak zřejmě nelze eliminovat z $\mathcal{G}$ všechna ε-pravidla.

Definice 3.13. Řekneme, že CFG $\mathcal{G} = (N, \Sigma, P, S)$ je bez *ε-pravidel* $\overset{def}{\Longleftrightarrow}$ buď

1. P neobsahuje žádné ε-pravidlo (tj. pravidlo tvaru $A \to \varepsilon$) nebo
2. v P existuje právě jedno ε-pravidlo $S \to \varepsilon$ a S se nevyskytuje na pravé straně žádného pravidla z P.

Algoritmus 3.4 Eliminace ε-pravidel

Vstup: CFG $\mathcal{G} = (N, \Sigma, P, S)$
Výstup: CFG $\mathcal{G}' = (N', \Sigma, P', S')$ bez ε-pravidel: $L(\mathcal{G}) = L(\mathcal{G}')$

Zkonstruuj $N_\varepsilon = \{A \in N | A \Rightarrow^* \varepsilon\}$ (* analogicky jako N_e z algoritmu 3.1 *); (1)
Množinu pravidel P' zkonstruuj takto:
for all $A \to X_1 \dots X_n \in P$ **do**
  přidej do P' všechna pravidla tvaru $A \to \alpha_1 \dots \alpha_n$ z P splňující tyto podmínky: (2)
    (a) pokud $X_i \notin N_\varepsilon$ (*tj. $X_i \not\Rightarrow^* \varepsilon$*), pak $\alpha_i = X_i$; (3)
    (b) pokud $X_i \in N_\varepsilon$ (*tj. $X_i \Rightarrow^* \varepsilon$*), pak α_i je buď X_i, nebo ε; (4)
    (c) ne všechna α_i jsou ε; (* tj. nepřidávej pravidlo $A \to \varepsilon$ *) (5)
end for
if $S \in N_\varepsilon$ (6)
  then přidej do P' pravidla $S' \to S|\varepsilon$ ($S' \notin N \cup \Sigma$); $N' := N \cup \{S'\}$ (7)
  else $N' := N$; $S' := S$ (8)

Věta 3.14. *Výstupní CFG $\mathcal{G}'$ z algoritmu 3.4 je bez ε-pravidel a $L(\mathcal{G}) = L(\mathcal{G})'$.*

Důkaz. Snadno se nahlédne, že $\mathcal{G}'$ je bez ε-pravidel – viz řádek (5). Abychom ukázali, že $L(\mathcal{G}) = L(\mathcal{G})'$, lze ukázat, že $A \Rightarrow^* w$ v $\mathcal{G}' \iff w \neq \varepsilon \wedge A \Rightarrow^* w$ v $\mathcal{G}$ Důkaz, který je ponechán čtenáři do cvičení, se vede indukcí vzhledem k délce derivace $A \overset{i}{\Rightarrow} w$ v $\mathcal{G}$ pro část '$\Longrightarrow$'; analogicky pro obrácenou implikaci. Požadovanou jazykovou ekvivalenci pro neprázdná slova obdržíme položením $A = S$ ve výše uvedeném tvrzení; fakt, že $\varepsilon \in L(\mathcal{G}) \iff \varepsilon \in L(\mathcal{G})'$ je zřejmý z řádků (6) – (8). □

Další užitečnou transformací může být odstranění pravidel $A \to B$, $(A, B \in N)$, která nazýváme *jednoduchá* pravidla.

Algoritmus 3.5 Eliminace jednoduchých pravidel

Vstup: CFG $\mathcal{G} = (N, \Sigma, P, S)$ bez ε-pravidel
Výstup: CFG $\mathcal{G}' = (N, \Sigma, P', S)$ bez jednoduchých a ε-pravidel: $L(\mathcal{G}) = L(\mathcal{G}')$

for all $A \in N$ **do**
  zkonstruuj $N_A = \{B \in N | A \Rightarrow^* B\}$ takto (* opět: srv. s $V_0, V_1, \dots$ z alg. 3.2*):
  $i := 0;\ N_i := \{A\}$; (* inicializace: reflexivita $\Rightarrow^*$ *) (1)
  repeat
    $i := i + 1$; (* iterace: transitivita $\Rightarrow^*$ *) (2)
    $N_i := N_{i-1} \cup \{C | B \to C \in P, B \in N_{i-1}\}$ (3)
  until $N_i = N_{i-1}$; (* test ukončení *) (4)
  $N_A := N_i$; (5)
end for
Množinu pravidel P' konstruuj takto: $P' := \emptyset$;
for all $B \to \alpha \in P$, které není jednoduché **do**
  přidej do P' pravidla $A \to \alpha$ pro všechna A taková, že $B \in N_A$ (6)
end for

Příklad 3.15. *Mějme gramatiku $\mathcal{G}_0$ z příkladu 3.2 s pravidly:*

$$
\begin{aligned}
E &\to E + T \mid T \\
T &\to T * F \mid F \\
F &\to (E) \mid i
\end{aligned}
$$

Po skončení 1. cyklu (řádky (1) – (5)) dostaneme $N_E = \{E, T, F\}$, $N_T = \{T, F\}$, $N_F = \{F\}$, což po skončení 2. cyklu (ř. (6)) dává výstupní gramatiku bez jednoduchých pravidel:

$$
\begin{aligned}
E &\to E + T \mid T * F \mid (E) \mid i \\
T &\to T * F \mid (E) \mid i \\
F &\to (E) \mid i
\end{aligned}
$$

Věta 3.16. *Gramatika $\mathcal{G}'$ z algoritmu 3.5 je bez jednoduchých pravidel a $L(\mathcal{G}) = L(\mathcal{G})'$.*

Důkaz. Množina pravidel P' je evidentně konstruována tak, že neobsahuje jednoduchá pravidla – viz řádek (6).

Nejprve ukažme, že $L(\mathcal{G})' \subseteq L(\mathcal{G})$. Mějme tedy $w \in L(\mathcal{G})'$, tedy v $\mathcal{G}'$ existuje derivace $S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \dots \Rightarrow \alpha_n = w$. Bylo-li při kroku $\alpha_i \Rightarrow \alpha_{i+1}$ použito v $\mathcal{G}'$

pravidlo $A \to \beta$, pak existuje nějaké $B \in N_A$ (s možností $A = B$) takové, že v $\mathcal{G}$ je $A \Rightarrow^* B$ a $B \Rightarrow \beta$, a tedy i $A \Rightarrow^* \beta$ a $\alpha_i \Rightarrow^* \alpha_{i+1}$ v $\mathcal{G}$. Odtud již snadno dostaneme, že v $\mathcal{G}$ existuje derivace $S \Rightarrow^* w$, tj. $w \in L(\mathcal{G})$.

Abychom ukázali platnost obrácené inkluse, tj. $L(\mathcal{G}) \subseteq L(\mathcal{G})'$, zvolme libovolné $w \in L(\mathcal{G})$. Pak v $\mathcal{G}$ existuje levá derivace $S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n = w$. Pak lze nalézt posloupnost indexů $i_1, \ldots, i_k$ složenou výhradně z j takových, že v $\alpha_{j-1} \Rightarrow \alpha_j$ nebylo použito jednoduchého pravidla (zejména derivace věty nemůže končit jednoduchým pravidlem, a proto $i_k = n$). Protože se jedná o levou derivaci, pak opakované použití jednoduchých pravidel nahrazuje neterminály na téže pozici v uvažované levé větné formě. Odtud vidíme, že v $\mathcal{G}'$ je možná derivace $S = \alpha_0 \Rightarrow \alpha_{i_1} \Rightarrow \ldots \Rightarrow \alpha_{i_k} = w$, tj. $w \in L(\mathcal{G})'$.

Celkem tedy máme žádané $L(\mathcal{G})' = L(\mathcal{G})$. □

Definice 3.17. CFG $\mathcal{G} = (N, \Sigma, P, S)$ se nazývá *necyklická*, právě když neexistuje $A \in N$ takový, že $A \Rightarrow^+ A$. $\mathcal{G}$ se nazývá *vlastní*, právě když je bez nepoužitelných symbolů, bez ε-pravidel a necyklická. *A-pravidlem* nazveme každé pravidlo tvaru $A \to \alpha$.

Věta 3.18. *Ke každému CFL existuje vlastní CFG, která jej generuje.*

Důkaz. Použitím výše uvedených algoritmů 3.3, 3.4 a 3.5 a odpovídajích tvrzení o jejich korektnosti. □

V dalším textu předpokládáme, že každá CFG je bez nepoužitelných symbolů a pokud nebude řečeno jinak, pak i vlastní.

3.1.3 Chomského normální forma, lemma o vkládání

V této části nejprve ukážeme, že ke každé CFG existuje ekvivalení CFG v jistém speciálním tvaru, který je charakterizován zejména tím, že na pravých stranách pravidel vystačíme se dvěma výskyty neterminálů (přesná definice viz 3.19). Na základě tohoto tvrzení budeme schopni dokázat tzv. lemma o vkládání (obecně též známé jako pumping lemma) pro CFL, které nám umožní v některých případech dokázat, že daný jazyk není CFL.

Definice 3.19. Řekneme, že CFG $\mathcal{G} = (N, \Sigma, P, S)$ je v *Chomského normální formě* (CNF) $\overset{def}{\Longleftrightarrow}$ $\mathcal{G}$ je bez ε-pravidel (viz def. 3.13) a každé pravidlo z P (s eventuelní výjimkou $S \to \varepsilon$) má jeden z těchto tvarů:

1. $A \to BC$, kde $B, C \in N$ nebo
2. $A \to a$, kde $a \in \Sigma$.

K důkazu tvrzení, že každý CFL je generovatelný nějakou CFG v CNF použijeme algoritmu 3.6; k důkazu jeho korektnosti využijeme následující lemma o substituci.

Lemma 3.20. ***(o substituci)*** *Nechť $\mathcal{G} = (N, \Sigma, P, S)$ je CFG. Nechť $A \to \alpha_1 B \alpha_2 \in P$ je pravidlo a $B \to \beta_1 \,|\, \ldots |\beta_r$ jsou všechna B-pravidla z P. Nechť dále $\mathcal{G}_1 = (N, \Sigma, P_1, S)$, kde $P_1 = (P \setminus \{A \to \alpha_1 B \alpha_2\}) \cup \{A \to \alpha_1 \beta_1 \alpha_2 \,|\, \ldots |\alpha_1 \beta_r \alpha_2\}$. Pak $L(\mathcal{G}) = L(\mathcal{G})_1$.*

Důkaz. Zřejmě $L(\mathcal{G})_1 \subseteq L(\mathcal{G})$, protože pokud je v derivaci v $\mathcal{G}_1$ použito nějaké pravidlo $A \to \alpha_1 \beta_i \alpha_2$, pak v $\mathcal{G}$ lze použít $A \Rightarrow \alpha_1 B \alpha_2 \Rightarrow \alpha_1 \beta_i \alpha_2$. K důkazu $L(\mathcal{G}) \subseteq L(\mathcal{G})_1$ stačí uvědomit si, že $A \to \alpha_1 B \alpha_2$ je jediné pravidlo, které je v $\mathcal{G}$ a není v $\mathcal{G}_1$. Tedy kdykoli

Algoritmus 3.6 Transformace do CNF

Vstup: Vlastní CFG $\mathcal{G} = (N, \Sigma, P, S)$ bez jednoduchých pravidel
Výstup: CFG $\mathcal{G}' = (N', \Sigma, P', S)$ v CNF: $L(\mathcal{G}) = L(\mathcal{G}')$

P' je tvořena takto: $P' := \emptyset$;
for all $p \in P$ **do**
  if pravidlo p je tvaru $A \to a$ nebo $A \to BC$ nebo $S \to \varepsilon$ **then** přidej p do P'; (1)
  if $p = A \to X_1X_2$, kde aspoň jedno z $X_i (i = 1, 2)$ je terminál **then** (2)
    polož $X_i' \stackrel{def}{=} \begin{cases} X_i, \text{ je-li } X_i \in N \\ \text{nový neterminál, je-li } X_i \in \Sigma \end{cases}$ (3)
    a do P' přidej pravidlo $A \to X_1'X_2'$; (4)
  if $p = A \to X_1 \dots X_k, k > 2$ **then** (5)
    polož $X_i' \stackrel{def}{=} \begin{cases} X_i, \text{ je-li } X_i \in N \\ \text{nový neterminál, je-li } X_i \in \Sigma \end{cases} \quad (1 \leq i \leq k)$ (6)
    a do P' přidej pravidla (7)

$$\begin{aligned} A &\to X_1'\langle X_2 \dots X_k\rangle \\ \langle X_2 \dots X_k\rangle &\to X_2'\langle X_3 \dots X_k\rangle \\ &\vdots \\ \langle X_{k-2} \dots X_k\rangle &\to X_{k-2}'\langle X_{k-1}X_k\rangle \\ \langle X_{k-1}X_k\rangle &\to X_{k-1}'X_k' \end{aligned}$$

    kde každé $\langle X_i \dots X_k\rangle$ je nový neterminál
end for
for all neterminál tvaru a' nově zavedený v ř. (3) nebo (6) **do**
  do P' přidej pravidla $a' \to a$ (8)
end for
$N' := N \cup \{$všechny nově zavedené neterminály tvaru a' nebo tvaru $\langle X_i \dots X_k\rangle\}$ (9)

je v nějaké derivaci věty v $\mathcal{G}$ toto pravidlo použito, pak neterminál B musí být přepsán v některém z pozdějších kroků derivace v $\mathcal{G}_1$ pomocí nějakého B-pravidla, tj. $B \to \beta_i$. Tyto dva kroky odvození v $\mathcal{G}$ lze v gramatice $\mathcal{G}_1$ nahradit jedním krokem $A \Rightarrow \alpha_1\beta_i\alpha_2$. □

Věta 3.21. *Nechť L je CFL. Pak $L = L(\mathcal{G})$ pro nějakou CFG $\mathcal{G}$ v CNF.*

Důkaz. Bez újmy na obecnosti předpokládejme, že L je generován nějakou CFG $\mathcal{G}$, která je vlastní a bez jednoduchých pravidel, a tedy splňuje požadavky kladené na vstupní gramatiku algoritmu 3.6. Inspekcí tohoto algoritmu snadno zjistíme, že výstupní gramatika $\mathcal{G}'$ splňuje požadavky CNF.

Zbývá tedy ukázat, že $L(\mathcal{G}) = L(\mathcal{G}')$. Opakovaně použijme lemma o substituci (viz 3.20) nejprve na každé pravidlo v $\mathcal{G}'$ obsahující nově zavedený neterminál a' a následně též na každé pravidlo s neterminálem tvaru $\langle X_i \dots X_k\rangle$. Takto obdržíme původní gramatiku $\mathcal{G}$. Vzhledem k tomu, že jsme opakovaně použili pouze lemma o substituci, které zachovává ekvivalenci gramatik, pak tedy i $\mathcal{G}$ a $\mathcal{G}'$ jsou ekvivaletní. □

Příklad 3.22. *Mějme $\mathcal{G}$ s pravidly* $S \to aAB \mid BA$
$A \to BBB \mid a$
$B \to aS \mid AS \mid b$

Do množiny pravidel P' hledané gramatiky v CNF nejprve přidáme pravidla, která již požadavky CNF splňují, tj. $S \to BA,\ A \to a,\ B \to AS \mid b$ (viz řádek (1) v algoritmu 3.6) V dalším kroku (viz ř. (2)) procházíme pravidla s pravou stranou délky 2 obsahující alespoň jeden terminál: do P' tedy přidáme $B \to a'S,$ kde a' je nový neterminál. Po tomto kroku máme zpracována všechna pravidla s délkou pravé strany nejvýše 2.

Následující krok (viz řádky (5) – (7)) prochází pravidla s délkami pravých stran většími než 2 a do P' tedy díky pravidlu $S \to aAB$ postupně přidáme $S \to a'\langle AB\rangle,\ \langle AB\rangle \to AB$ a díky pravidlu $A \to BBB$ přidáme $A \to B\langle BB\rangle, \langle BB\rangle \to BB$.

Konečně v kroku (8) do P' přidáme pro všechny nově zavedené "čárkované" neterminály pravidla, která je přepisují na původní terminály, tj. do P' přidáme $a' \to a$.

Poznámka 3.23. *(O derivačních stromech gramatik v CNF) Uvědomme si, že (i) z každého uzlu derivačního stromu gramatiky v CNF vychází nejvýše 2 hrany a (ii) z uzlu vychází jedna hrana právě když tato vchází do listu – pokud bychom z takového stromu odstranili listy (a hrany do nich vcházející), obdrželi bychom binární strom, v němž platí, že pokud každá cesta má délku (tj. počet hran na této cestě) rovnu j, pak strom má 2^j listů. Pro stromy odvození v CNF je však počet listů roven počtu jejich přímých předchůdců – viz (ii). Pokud tedy strom v CNF nemá žádnou cestu delší než j, pak má nejvýše 2^{j-1} listů.*

Věta 3.24. ***(Lemma o vkládání, pumping lemma pro CFL)*** *Nechť L je CFL. Pak existují přirozená čísla p, q (závisející na L) taková, že každé slovo $z \in L,\ |z| > p$ lze psát ve tvaru $z = uvwxy,$ kde*

- *alespoň jedno ze slov v, x je neprázdné (tj. $vx \neq \varepsilon$),*
- *$|vwx| \leq q$ a*
- *$uv^iwx^iy \in L$ pro všechna $i \geq 0$.*

Idea důkazu: Mějme $\mathcal{G}$ v CNF. Díky poznámce 3.23 víme, že pokud zvolíme slovo z "dostatečně dlouhé", pak v derivačním stromu musí být "dost dlouhá" cesta. Zvolme tedy z tak dlouhé, aby v jeho derivačním stromu byla tak dlouhá cesta, že se na ní některý (tj. alespoň jeden) neterminál, řekněme A, musí vyskytnout alespoň dvakrát (viz též obrázek 3.1).

Tedy A se musí (díky výskytu, který je na zmíněné cestě blíže k listu) přepsat na nějaký terminální řetěz, řekněme w (tj. $A \Rightarrow^* w$). Rovněž však se A musí (díky výskytu, který je na téže cestě blíže ke kořenu) přepsat na řetěz opět obsahující sebe sama, řekněme vAx (tj. $A \Rightarrow^* vAx$); toto přepisování můžeme libovolněkrát opakovat (též i 0-krát). Vždy obdržíme korektní odvození nějakého slova z $L(\mathcal{G})$. Neprázdnost alespoň jednoho z řetězů v či x je zajištěna tím, že CFG v CNF nemá ε-pravidla.

Konečně si uvědomme, že na dosti dlouhé cestě se může vyskytnout celá řada několikrát se opakujících neterminálů. Můžeme však A zvolit tak, aby oba jeho výše zmíněné výskyty nebyly "příliš daleko" od listu, tj. tak, aby úsek vwx nebyl příliš dlouhý.

Důkaz. Nechť L je generován nějakou CFG $\mathcal{G} = (N, \Sigma, P, S)$, která je (bez újmy na obecnosti) v CNF. Označme $k = card(N)$ a položme $p = 2^{k-1}, q = 2^k$.

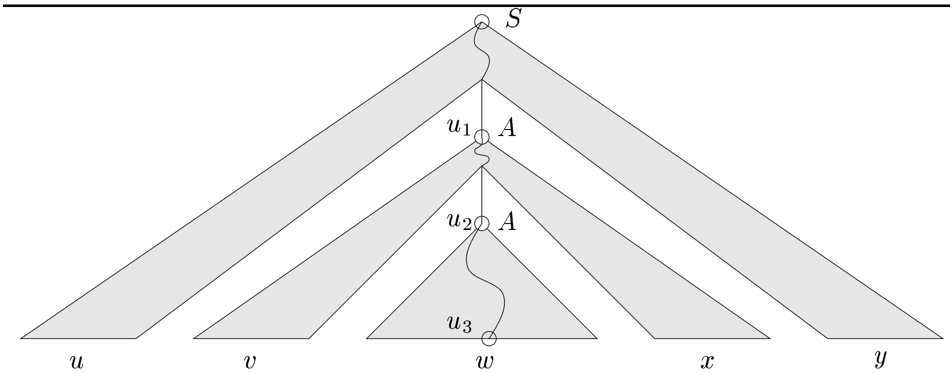
Je-li $z \in L, |z| > p$, pak v libovolném derivačním stromu slova z existuje cesta délky větší než k – viz poznámka 3.23 o derivačních stromech gramatik v CNF. Zvolme pevně jeden takový derivační strom T a v něm (libovolnou) nejdelší cestu C, která má jistě délku větší než k. Na této cestě C lze zvolit tři uzly u_1, u_2, u_3 s těmito vlastnostmi:

1. uzly u_1, u_2 jsou označeny týmž neterminálem, řekněme A,
2. u_1 leží blíže ke kořenu než u_2,
3. u_3 je list a
4. cesta z u_1 do u_3 má délku nejvýše $k + 1$.

Uzel u_1 určuje v T podstrom T_1 s kořenem u_1, tedy výsledek z_1 podstromu T_1 je podslovem ve slově z, které je výsledkem stromu T; tj. existuje derivace

$S \Rightarrow^* uAy$ pro nějaká u, y. (1)

Podobně u_2 určuje v T podstrom T_2 s kořenem u_2, přičemž T_2 je podstromem stromu T_1 – viz obrázek 3.1.



Obrázek 3.1: Derivační strom s cestou C se dvěma výskyty neterminálu A

Strom T_1 odpovídá (levé) derivaci nějakého slova z_1, přičemž z_1 má délku nejvýše 2^k (viz vlastnost 4 a pozn.3.23 – každá cesta v T_1 má délku nejvýše $k + 1$: kdyby v T_1 existovala od u_1 k nějakému u_4 cesta delší, pak by i v T byla cesta delší než je zvolená C, a to by byl spor s předpokladem o volbě C jako cesty s největší délkou). Jistě tedy platí $|z_1| \leq 2^k = q$.

Strom T_2 též odpovídá derivaci nějakého slova, označme jej w. Jelikož T_2 je podstromem ve stromu T_1, musí být w podslovem slova z_1, tj. existují řetězy v, x takové, že lze psát $z_1 = vwx$, což spolu s již ukázaným $|z_1| \leq 2^k = q$ dává žádané $|vwx| \leq q$. Dále si uvědomme, že u_1 je vnitřním uzlem, a tedy mu odpovídá aplikace pravidla tvaru $A \rightarrow BC$, a tedy alespoň jedno ze slov v, x je neprázdné (CFG v CNF je bez ε-pravidel), tj. $vx \neq \varepsilon$. Současně jsme též ukázali, že

$A \Rightarrow^* vAx$ a (2)

$A \Rightarrow^* w$ (3)

Nyní použijme jedenkrát derivaci (1), pak i-krát ($i \geq 0$) derivaci (2) a nakonec derivaci (3), tj.

$S \Rightarrow^* uAy \Rightarrow^* uv^iAx^iy \Rightarrow^* uv^iwx^iy$

tedy $uv^iwx^iy \in L(\mathcal{G})$.

Všimněme si, že derivace ad (2) umožňuje náhradu, kdy v T místo podstromu T_2 opakovaně (i-krát) vložíme podstrom T_1 s kořenem označeným týmž A – takto získáme opět korektní derivační strom; derivace (3) umožňuje (mimo jiné) místo T_1 použít přímo T_2 – odpovídá situaci pro $i = 0$. □

Explicitněji (z hlediska kvantifikace) lze tvrzení lemmatu 3.24 přepsat takto:

$$\begin{array}{l} \text{(Nechť) } L \text{ je CFL} \implies \exists p, q \in \mathbb{N}. \\ \qquad \forall z \in L.\ |z| > p\,. \\ \qquad\quad \exists\, u, v, w, x, y.\ z = uvwxy \land \\ \qquad\qquad vx \neq \varepsilon \land \\ \qquad\qquad |vwx| \leq q\,. \\ \qquad\quad \forall i \geq 0.\ uv^i wx^i y \in L \end{array}$$

Uvědomme si, že lemma o vkládání je (opět) pouze podmínkou *nutnou* k tomu, aby L byl CFL. Zopakujme, že i toto lemma je – stejně jako PL pro regulární jazyky – tvaru implikace $P \implies Q$, kde P je nyní výrok, že L je CFL a Q jsou uvedené vlastnosti; této implikace (resp. v kontrapositivní formě ekvivaletní implikace $\neg Q \implies \neg P$), lze použít k důkazu, že nějaký jazyk L *není* CFL, nikoli však obráceně, tj. k důkazu, že L je CFL. Čtenáři doporučejeme, aby si $\neg Q$ explicitně vyjádřil.

Poznámka 3.25. *Všimněme si, že pokud ve výše uvedeném lemmatu 3.24 namísto konstant p, q budeme všude psát jen (jedinou) pumpovací konstantu n, tvrzení zůstane v platnosti: v důkazu stačí položit $p = q = 2^k$, kde $k = card(N)$.*

Příklad 3.26. *Ukažme, že jazyk $L = \{a^i b^i c^i \mid i \geq 1\}$ není CFL. Náš postup bude tento: předpokládejme opak, tj. $P \equiv L$ je CFL a dokazujme $\neg Q \implies \neg P$.*

1. *Nechť n je libovolná konstanta z poznámky 3.25 (či první konstanta z lemmatu 3.24 a $p = q = n$).*
2. *Zvolme nyní slovo z (v závislosti na n – pro každé n musí existovat takové z) a*
3. *uvážíme všechny možnosti, jak jej zapsat jako $z = uvwxy$ tak, aby splňovalo podmínky lemmatu ($vx \neq \varepsilon, |vwx| \leq n$);*
4. *následně pak ukažme, že pro každé takové rozdělení z na u, v, w, x, y lze nalézt (existuje) takové i, že napumpováním získáme $uv^i wx^i y \notin L$.*

Dle 1 nechť n je libovolné; zvolme (viz 2) $z = a^n b^n c^n$ a dále postupujeme podle 3 a 4.

Pokud by v obsahovalo kladný počet symbolů a a kladný počet symbolů b, pak napumpované $uv^2wx^2y \notin L$, protože po nějakém výskytu b by následoval výskyt a, tj. $uv^2wx^2y \notin a^*b^*c^*$. Podobně postupujeme ve všech ostatních případech, kdy v nebo x obsahují nenulový počet výskytů (alespoň) 2 různých znaků.

Uvažme proto zbývající možnosti, kdy v patří do a^* nebo b^* nebo c^* a též x obsahuje jen samé stejné znaky, přičemž alespoň jedno ze slov v, x je neprázdné. Je-li $v \in a^+$ a $x \in b^*$, pak uv^2wx^2y obsahuje více symbolů a než symbolů c, tj. $uv^2wx^2y \notin L$. Podobně dojdeme ke sporu při všech ostatních možných volbách: protože vx obsahuje aspoň jeden výskyt symbolu d s možností výskytu jiného symbolu e, nezbývá žádná možnost pro výskyt třetího symbolu f (pro $d, e, f \in \{a, b, c\}$ vzájemně různé), a tedy uv^2wx^2y bude mít vždy více výskytů symbolu d než výskytů symbolu f.

Jak již víme, pro porozumění formuli s větším počtem kvantifikátorů je vhodné na ni nahlížet jako na hru dvou hráčů (viz též pumping lemma pro regulární jazyky), kde kvantifikátoru $\forall$ odpovídá hráč Al a $\exists$ je reprezentován hráčem Ex. Nechť tedy L je jazyk, pak hra (ve variantě $p = n = q$) probíhá takto:

- Ex zvolí přirozené číslo n
- Al zvolí $z \in L$ takové, že $|z| \geq n$
- Ex zvolí u, v, w, x, y tak, aby platilo $z = uvwxy,\ |vwx| \leq n$ a $vx \neq \varepsilon$
- Al zvolí i

Pokud Al zvolí i tak, že $uv^iwx^iy \notin L$, pak vyhrává Al. Pokud Al může vždy vyhrát bez ohledu na to, jak hraje Ex (má vyhrávající strategii), pak L není CFL. Pokud však Al prohraje (ať už vždy či jen někdy), pak z pumping lemmatu nelze o L odvodit nic korektního.

Příklad 3.27. *Jiným příkladem jazyka, který není CFL, je $L = \{a^ib^jc^id^j \mid i, j \geq 0\}$. Abychom to ukázali, předpokládejme, že L je CFL. Jelikož chceme dospět ke sporu, hrajeme roli hráče Al.*

- *Hráč Ex zvolí libovolnou pumpovací konstantu n;*
- *dále Al zvolí nějaké $z = a^nb^nc^nd^n$.*
- *Dle pumping lemmatu tedy mají existovat u, v, w, x, y tak, že $z = uvwxy,\ |vwx| \leq n$ a v nebo x je neprázdné (volí Ex).*
- *Má platit $uv^iwx^iy \in L$ pro všechna $i \geq 0$. Al tedy má ukázat, že pro každou Ex-ovu volbu* vyvrátí *predikát $\forall i.\ uv^iwx^iy \in L$):*
  Jelikož $|vwx| \leq n,\ vwx$ musí být tvaru a^*b^* nebo b^*c^* nebo c^*d^*. Pro jakoukoli z těchto voleb však "pumpování" musí vyústit v řetěz nepatřící do L (např. $vwx \in a^*b^*$ značí, že uwy má méně symbolů a a b, než symbolů c a d).

K důkazu, že například $L = \{a^ib^jc^k \mid i \neq j \text{ a } j = k\}$ nebo $L = \{a^ib^jc^k \mid i \neq j \text{ a } j \neq k\}$ není CFL je nutno použít některou ze silnějších variant Pumping lemmatu pro CFL, například tzv. Ogdenovo lemma, resp. jeho důsledky. Dokonce existuje též nutná a postačující podmínka pro to, aby jazyk byl CFL. Čtenáře odkažme na seznam literatury uvedený v závěru.

3.1.4 Greibachové normální forma

Naším cílem je nyní prezentovat další, tzv. Greibachové normální formu (GNF), v níž každá pravá strana každého pravidla v CFG začíná terminálním symbolem (za nímž případně následují neterminály). Abychom mohli dokázat tvrzení o existenci ekvivaletní gramatiky v GNF, potřebujeme ukázat několik lemmat o modifikacích pravidel v CFG.

Definice 3.28. Neterminál A v CFG $\mathcal{G} = (N, \Sigma, P, S)$ se nazývá *rekursivní* jestliže existuje derivace $A \Rightarrow^+ \alpha A \beta$ pro nějaká α, β. Je-li $\alpha = \varepsilon$ (resp. $\beta = \varepsilon$), pak A se navývá *levorekursivní* (resp. *pravorekursivní*). CFG bez levorekursivních neterminálů se nazývá nelevorekursivní.

Poznamenejme, že pojem rekursivity neterminálu je diametrálně odlišný od pojmu rekursivity gramatiky, tj. existence algoritmu pro rozhodování problému zda $w \in L(\mathcal{G})$.

Pro transformaci CFG do GNF bude nutné z gramatiky odstranit levou rekursi: pokud v gramatice nebudou levorekursivní neterminály, budeme moci na nejlevější neterminály na pravých stranách pravidel aplikovat lemma o substituci. Schopnost eliminovat levou rekursi se ukáže jako velmi užitečná též v řadě prakticky orientovaných aplikací.

Lemma 3.29. *(**O eliminaci bezprostřední levé rekurze**). Nechť $\mathcal{G} = (N, \Sigma, P, S)$ je CFG, v níž všechna A-pravidla jsou tvaru*

$A \to A\alpha_1 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \ldots \mid \beta_n,$ *kde* $1 : \beta_i \neq A$ *pro všechna* $1 \leq i \leq n.$ *(*)*

Nechť $\mathcal{G}' = (N \cup \{A'\}, \Sigma, P', S)$, kde P' obdržíme z P tak, že všechna pravidla označená (*) nahradíme pravidly

$A \to \beta_1 \mid \ldots \mid \beta_n \mid \beta_1 A' \mid \ldots \mid \beta_n A', \quad A' \to \alpha_1 \mid \ldots \mid \alpha_m \mid \alpha_1 A' \mid \ldots \mid \alpha_m A'.$ *(**)*

Pak $L(\mathcal{G}) = L(\mathcal{G}')$.

Důkaz. Neformálně řečeno jsme levou rekursi v A-pravidlech, která generují řetězce tvaru $(\beta_1 + \ldots + \beta_n)(\alpha_1 + \ldots + \alpha_m)^*$, nahradili nově zavedeným pravorekursivním A', který zajistí, že A generuje tutéž množinu. Formální důkaz možno vést například takto:
V libovolné nejlevější derivaci nějaké věty v $\mathcal{G}$ musí posloupnost použití pravidel typu $A \to A\alpha_i$ být ukončena použitím nějakého pravidla $A \to \beta_j$. Tedy místo derivace

$$A \Rightarrow A\alpha_{i_1} \Rightarrow A\alpha_{i_2}\alpha_{i_1} \Rightarrow \ldots \Rightarrow A\alpha_{i_p}\ldots\alpha_{i_2}\alpha_{i_1} \Rightarrow \beta_j\alpha_{i_p}\ldots\alpha_{i_2}\alpha_{i_1}$$

v $\mathcal{G}$, lze v gramatice $\mathcal{G}'$ použít derivaci

$$A \Rightarrow \beta_j A' \Rightarrow \beta_j\alpha_{i_p} A' \Rightarrow \ldots \Rightarrow \beta_j\alpha_{i_p}\ldots\alpha_{i_2} A' \Rightarrow \beta_j\alpha_{i_p}\ldots\alpha_{i_2}\alpha_{i_1}.$$

Jelikož výše uvedená úvaha platí i v obráceném směru (po jisté posloupnosti použití pravidel tvaru $A' \to \alpha_i A'$ musí být použito $A' \to \alpha_i$), dostáváme žádané $L(\mathcal{G}) = L(\mathcal{G}')$. □

Příklad 3.30. *Mějme $\mathcal{G}_0$ s pravidly:*

$$\begin{aligned} E &\to E{+}T \mid T \\ T &\to T * F \mid F \\ F &\to (E) \mid i \end{aligned}$$

Pak $\mathcal{G}_0'$ má pravidla:

$$\begin{array}{ll} E \to T \mid TE' & E' \to +T \mid +TE' \\ T \to F \mid FT' & T' \to *F \mid *FT' \\ F \to (E) \mid i & \end{array}$$

Poznámka 3.31. *Jestliže $\mathcal{G}$ z lemmatu 3.29 je bez ε-pravidel, pak i ekvivaletní $\mathcal{G}'$ má tuto vlastnost. Pravidla (*) lze též nahradit namísto pravidel (**) pravidly:*

$A \to \beta_1 A' \mid \ldots \mid \beta_n A', \quad A' \to \alpha_1 A' \mid \ldots \mid \alpha_m A' \mid \varepsilon.$ *(***)*

Tato náhrada však zavádí ε-pravidla: ke $\mathcal{G}_0$ z příkladu 3.30 bychom obrželi $\mathcal{G}_0''$ s pravidly:

$$\begin{array}{ll} E \to TE' & E' \to +TE' \mid \varepsilon \\ T \to FT' & T' \to *FT' \mid \varepsilon \\ F \to (E) \mid i & \end{array}$$

Lemma 3.29 nedává návod jak postupovat, když v $\mathcal{G}$ existuje levorekursivní smyčka $A \Rightarrow^+ A\alpha$ délky větší než 1. Tento problém řeší algoritmus 3.7.

Algoritmus 3.7 Eliminace levé rekurze

Vstup: Vlastní CFG $\mathcal{G} = (N, \Sigma, P, S)$
Výstup: CFG $\mathcal{G}' = (N', \Sigma, P', S) : L(\mathcal{G}) = L(\mathcal{G}')$
a $\mathcal{G}'$ je nelevorekurzivní

1: Uspořádej libovolně N, $N = \{A_1, \dots, A_n\}$
2: **for** $i := 1$ **to** n **do**
3:   **for** $j := 1$ **to** $i - 1$ **do**
4:     **for all** pravidlo tvaru $A_i \to A_j\alpha$ (*tj. zde platí $j < i$ *) **do**
5:       přidej $A_i \to \beta_1\alpha|\dots|\beta_k\alpha$, kde $A_j \to \beta_1|\dots|\beta_k$ jsou všechna A_j-pravidla;
6:       vypusť pravidlo $A_i \to A_j\alpha$ (* = aplikace lemmatu o substituci *)
7:     **end for**
8:   **end for**
9:   (* dále následuje použití eliminace bezprostřední levé rekurze pro A_i-pravidla: *)
10:   **for all** pravidlo tvaru $A_i \to A_i\alpha$ **do**
11:     přidej $A_i' \to \alpha A_i' \mid \varepsilon$; (* tj. použití pravidel (***); variantu (**) lze též použít *)
12:     vypusť pravidlo $A_i \to A_i\alpha$
13:   **end for**
14:   **for all** $A_i \to \beta_1|\dots|\beta_l$ taková, že $1 : \beta_k \neq A_i$ **do**
15:     přidej $A_i \to \beta_1 A_i'|\dots|\beta_l A_i'$,
16:   **end for**
17: **end for**

Věta 3.32. *Každý CFL je generovatelný nelevorekursivní CFG.*

Důkaz. Nechť $\mathcal{G}$ je vlastní CFG generující L. Jelikož algoritmus 3.7 používá jen transformace uvedené v lemmatu o substituci (3.20) a lemmatu o eliminaci bezprostřední levé rekurze (3.29), které zachovávají ekvivalenci gramatik, pak použití tohoto algoritmu na $\mathcal{G}$ dává ekvivaletní CFG $\mathcal{G}'$.

Zbývá tedy ukázat, že výsledná $\mathcal{G}'$ je bez levé rekurze. Indukcí ukažme platnost následujících dvou tvrzení:

1. po skončení i-té iterace vnějšího cyklu (začínajícího na řádku 2) platí, že všechna A_i-pravidla začínají buď terminálem nebo neterminálem $A_k, k > i$
2. po skončení j-té iterace vnitřního cyklu (začínajícího na řádku 3) a dané i platí, že A_i-pravidla začínají buď terminálem nebo neterminálem $A_k, k > j$.

Tuto indukci povedeme vzhledem k hodnotě $s = n \cdot i + j$, kde $i \in \langle 0, n\rangle, j \in \langle 0, i - 1\rangle$.

Báze indukce je pro $s = n$, tj. $i = 1, j = 0$ a odpovídá provedení eliminace levé rekurze pro A_1 (cyklus pro j se vůbec neprovedl); žádné $\beta_1, \dots, \beta_l$ nezačíná A_1. Tedy pro $i = 1$ tvrzení 1 platí; tvrzení 2 platí triviálně.

Indukční krok: předpokládejme, že tvrzení 1 a 2 platí pro všechny hodnoty menší než s a nechť nyní jsou i, j taková, že $0 \leq j < i \leq n$ a $n \cdot i + j = s$.
Dokažme nejprve 2. Na základě indukčního předpokladu o tvrzení 1 máme, že všechna A_j-pravidla začínají buď terminálem nebo neterminálem $A_k, k > j$ (je-li totiž $j > 1$, pak

instance 2 pro i a $j-1$ má hodnotu menší než s; případ $j=1$ plyne z 1). Tvrzení 2 pro parametry i a j tak okamžitě plyne z tvaru nově přidaných pravidel.
Induktivní krok pro tvrzení 1 a hodnotu s (tj. $n \cdot i = s, j = 0$) je ponechán čtenáři.

Z právě dokázaného tvrzení 1 plyne, že žádné z $A_1, \dots A_n$ nemůže být levorekursivní: kdyby totiž bylo $A_i \Rightarrow^+ A_i\alpha$ pro nějaké α, pak by musely existovat neterminály A_j a $A_k, k \leq j$ takové, že $A_i \Rightarrow^* A_j\beta \Rightarrow A_k\gamma \Rightarrow^* A_i\alpha$. Zbývá ukázat, že žádný z A_i' není levorekursivní, což však plyne z předpokladu, že $\mathcal{G}$ je vlastní CFG – tj. žádné α z řádku 10 není ε, a tedy A_i' nemůže nikdy být nejlevějším symbolem na pravé straně přidávaného pravidla $A_i' \to \alpha A_i'$ (viz řádek 14). □

Definice 3.33. Nechť $\mathcal{G} = (N, \Sigma, P, S)$ je CFG. Řekneme, že $\mathcal{G}$ je v *Greibachové normální formě* (GNF) právě když $\mathcal{G}$ je bez ε-pravidel (viz def. 3.13) a každé pravidlo z P (s eventuelní výjimkou $S \to \varepsilon$) je tvaru $A \to a\alpha$ kde $a \in \Sigma, \alpha \in N^*$.

Algoritmus 3.8 Transformace do GNF

Vstup: Nelevorekursivní, vlastní CFG $\mathcal{G} = (N, \Sigma, P, S)$
Výstup: CFG $\mathcal{G}'$ v GNF: $L(\mathcal{G}) = L(\mathcal{G}')$

1: Zkonstruuj na N lineární uspořádání $\prec$ takové, že je-li $A \to B\alpha \in P$, pak $A \prec B$. Označme $N = \{A_1, \dots, A_n | \ A_{i-1} \prec A_i, 0 < i \leq n\}$
2: **for** $i := n-1$ **downto** 1 **do**
3:   **for all** pravidlo tvaru $A_i \to A_j\alpha, \ j > i$ **do**
4:     přidej $A_i \to \beta_1\alpha | \dots | \beta_k\alpha,$ kde $A_j \to \beta_1 | \dots | \beta_k$ jsou vš. A_j-pravidla;
5:     vypusť pravidlo $A_i \to A_j\alpha$ (* = aplikace lemmatu o substituci *)
6:   **end for**
7: **end for**
8: **for all** pravidlo tvaru $A_i \to aX_1 \dots X_k, \exists X_j . 1 \leq j \leq k : \ X_j \in \Sigma$ **do**
9:   v pravidle $A_i \to aX_1 \dots X_k$ nahraď všechny ty X_j, které jsou terminály, novými neterminály X_j' (přidej je do množiny neterminálů) a přidej pravidla $X_j' \to X_j$.
10: **end for**

Věta 3.34. *Nechť L je CFL. Pak $L = L(\mathcal{G})$ pro nějakou CFG $\mathcal{G}$ v GNF.*

Důkaz. Bez újmy na obecnosti lze předpokládat, že L je generován CFG $\mathcal{G}_1$ splňující vstupní předpoklady algoritmu 3.8 (aby $\mathcal{G}_1$ byla vlastní, tj. specielně bez ε-pravidel, stačí v algoritmu 3.7 uvažovat na řádku 11 variantu (**) - viz též poznámka 3.31).

Lineární uspořádání $\prec$ požadované v řádku 1 algoritmu 3.8 lze zkonstruovat například takto: na množině neterminálů definujme relaci R : $\langle A, B\rangle \in R \overset{def}{\Longleftrightarrow} A \Rightarrow^+ B\alpha$ pro nějaké α. Pak díky tomu, že $\mathcal{G}_1$ není levorekursivní, je R částečným uspořádáním na množině neterminálů N, a tedy N lze lineárně douspořádat tak, aby platilo $R \subseteq \prec$.

Nyní ukažme, že po skončení cyklu, který začíná na řádku 2, pro hodnotu i musí platit, že všechna A_i-pravidla začínají terminálem. Toto se snadno ukáže indukcí pro hodnotu $k = n - i, \ i = 0, \dots, n-1$ (stručně zvanou zpětná indukce pro i od n po 1).

Báze indukce je pro $k = n$, tj. $i = 0$. Pravé strany A_n-pravidel jistě začínají terminálem (viz definice $\prec$), což je též důvodem, proč se cyklus 2: pro $i = n$ vůbec neprovádí.

Indukční krok: jestliže dokazovanou vlastnost mají všechna A_j-pravidla pro $i < j \leq n$, pak evidentně pro provedení řádků 4 a 5 mají tuto vlastnost i všechna A_i-pravidla. Tedy po úplném skončení cyklu na řádcích 2–7 začínají všechna pravidla terminálem.

Konečně uvažme, že algoritmus používá jen transformace zachovávající ekvivalenci gramatik, a tedy dostáváme žádané. □

Příklad 3.35. *Mějme $\mathcal{G}_0'$ z příkladu 3.30 s pravidly:*

$$
\begin{array}{ll}
E \to T \mid TE' & E' \to +T \mid +TE' \\
T \to F \mid FT' & T' \to *F \mid *FT' \\
F \to (E) \mid i &
\end{array}
$$

Požadované lineární uspořádání (jedno z možných) je například: $E' \prec E \prec T' \prec T \prec F$.

Další práce algoritmu 3.8 na řádcích 2 – 7 vypadá takto:

- *V množině pravidel hledané gramatiky v GNF zůstanou beze změny obě F-pravidla (F je největším prvkem vzhledem k $\prec$ a cyklem pro i nejsou vůbec zpracovávány – pravé strany F-pravidel totiž už terminálem začínají).*
- *Při průchodu cyklem pro neterminál T pravidla $T \to F \mid FT'$ nahradíme pravidly $T \to (E) \mid i \mid (E)T' \mid iT'$.*
- *V dalším průchodu (pro T') neprovádíme žádné změny.*
- *Při zpracování E nahradíme existující E-pravidla novými pravidly $E \to (E) \mid i \mid (E)T' \mid iT' \mid (E)E' \mid iE' \mid (E)T'E' \mid iT'E'$;*
- *pro E' opět žádné náhrady neprovádíme.*

Konečně (viz řádky 8 – 10) ve všech takto získaných pravidlech nahradíme na jejich pravých stranách všechny výskyty terminálu "$)$" novým neterminálem "$)'$" a přidáme pravidlo $)' \to)$.

Poznamenejme, že výše uvedená metoda převodu libovolné CFG do GNF není jediná možná – další metody lze nalézt v publikacích uvedených v seznamu literatury. Námi uvedená metoda byla zvolena zejména z toho důvodu, že explicitně obsahuje algoritmus na odstranění levé rekurze, jejíž přítomnost může činit jisté problémy v některých praktických aplikacích.

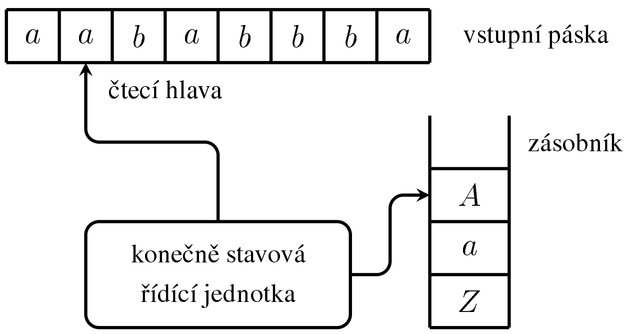
Mimo uvedenou GNF (viz definice 3.33) existují i její další varianty. Řekneme, že $\mathcal{G}$ je v *k-GNF* právě když je v GNF a žádná pravá strana v pravidlech gramatiky $\mathcal{G}$ nemá délku větší než k, $k \geq 3$, tj. neobsahuje více než $k - 1$ výskytů neterminálů (na intuitivní úrovni se jedná o GNF beroucí v potaz fakt, že díky CNF vystačíme v CFG na pravých stranách s (nejvýše) dvěma neterminály). Další variantou je tzv. *zdvojená GNF*, kdy všechny pravé strany pravidel jsou ze $\Sigma N^* \Sigma \cup \Sigma$, tj. začínají i končí terminálem. Konečně $\mathcal{G}$ je ve *zdvojené k-GNF* jestliže je ve zdvojené GNF a každé pravidlo má délku nejvýše k, $k \geq 4$. Například tedy 3-GNF či zdvojená 4-GNF neobsahují na pravých stranách pravidel více než 2 výskyty neterminálů. Algoritmy převodu libovolné CFG do každé z výše uvedených normálních forem lze nalézt v obsažnějších monografiích o bezkontextových jazycích.

Na závěr zmiňme ještě tzv. *operátorovou normální formu*, kdy na pravých stranách pravidel spolu nesmí sousedit žádné dva neterminály, tj. všechny pravé strany jsou ze $\Sigma^* N \Sigma^+ \dots \Sigma^+ N \Sigma^*$. Algoritmus převodu libovolné CFG (či bez újmy na obecnosti CFG v CNF) spočívá v zavedení nových neterminálů $\langle X, a\rangle$ pro každou dvojici $X \in N, a \in \Sigma$ s tím, že $\langle X, a\rangle$ má generovat množinu slov u takových, že X generuje slova ua. Tedy každý jazyk $L(X)$ je přesně sjednocením (přes $a \in \Sigma$) všech jazyků $L(\langle X, a\rangle).a$, případně doplněném o $\{\varepsilon\}$, pokud $L(X)$ obsahuje prázdné slovo. Tedy na pravých stranách pravidel lze každý neterminál X nahradit zřetězením $\langle X, a\rangle.a$; přidáním pravidel tvaru $\langle X, a\rangle \to \alpha$ pro každé $X \to \alpha a$, získáme ekvivaletní gramatiku v požadovaném tvaru. Detaily algoritmu i důkazu jeho korektnosti ponecháváme čtenáři jako cvičení.

Použití normálních forem pro další zkoumání vlastností CFL bylo již částečně ilustrováno v této části (viz CNF a její využití v důkazu pumping lemmatu pro CFL); s dalšími aplikacemi se setkáme v části 3.3.

3.2 Zásobníkové automaty

Tak jako k regulárním gramatikám existují konečné automaty, které rozpoznávají právě jazyky generované těmito gramatikami, tak i ke gramatikám bezkontextovým existují ve výše uvedeném smyslu ekvivaletní automaty – tzv. zásobníkové automaty (push-down automata – PDA). PDA si lze představit jako (nedeterministický) konečný automat s tím, že navíc obsahuje (pomocnou) paměť (a díky ní i další zdroj nedeterminismu), která pracuje jako zásobník (push-down store) a jejíž velikost není shora omezena – je tzv. *potenciálně* nekonečná v následujícím smyslu: v každém okamžiku (konečného) výpočtu je sice konečná (tj. v zásobníku je uloženo jen konečně mnoho symbolů), ale kdykoli ji můžeme rozšířit o další konečný počet paměťových míst (přidat další symboly na zásobník).



Obrázek 3.2: Zásobníkový automat

Ze vstupní pásky, na níž je zapsáno slovo nad jistou vstupní abecedou, lze pouze číst a čtecí hlava se pohybuje jen vpravo. Automat může na vrchol zásobníku ukládat symboly (opět z jisté abecedy) a takto uložené symboly může následně číst s tím, že smí číst pouze z vrcholu zásobníku: přečtený symbol je z vrcholu odstraněn (tj. systém LIFO – Last In

First Out). Jinými slovy, nelze číst do hloubi zásobníku, aniž by přečtené symboly nebyly odstraněny – zásobník, u něhož naopak toto "nedestruktivní" čtení lze realizovat se v angličtině nazývá *stack*, na rozdíl od našeho *push-down store* s desktruktivním čtením. Takto intuitivně popsané zařízení nyní formalizujme.

3.2.1 Definice PDA

Definice 3.36. *Nedeterministický zásobníkový automat* (PDA) je sedmice

$$\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F), \text{ kde}$$

- Q je konečná množina, jejíž prvky nazýváme *stavy*,
- Σ je konečná množina, tzv. *vstupní abeceda*,
- Γ je konečná množina, tzv. *zásobníková abeceda*,
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma) \to \mathcal{P}_{Fin}(Q \times \Gamma^*)$ je (parciální) *přechodová funkce*[2] ,
- $q_0 \in Q$ je *počáteční stav*,
- $Z_0 \in \Gamma$ je *počáteční symbol v zásobníku*,
- $F \subseteq Q$ je množina *koncových stavů*.

Je-li $\delta(p, a, Z)$ definováno, pak zápis $\delta(p, a, Z) = \{(q_i, \gamma_i) | \ 1 \leq i \leq n\}$ intuitivně interpretujeme tak, že PDA $\mathcal{M}$ může ze stavu p po přečtení symbolu a ze vstupní pásky a symbolu Z z vrcholu zásobníku (Z se při tomto čtení z vrcholu odstraní) přejít do jednoho ze stavů q_i a na vrchol zásobníku zapíše řetěz γ_i (přičemž na vrcholu zásobníku je nyní nejlevější symbol z γ_i). Čtecí hlava se na vstupní pásce posune o jeden symbol vpravo. Obdobně interpretujeme (pokud je definováno) $\delta(p, \varepsilon, Z)$ s tím rozdílem, že pozice čtecí hlavy na vstupní pásce se nemění.

Tuto intuitivní představu o jednom kroku výpočtu budeme formalizovat (obdobně jako u konečných automatů) zavedením pojmu konfigurace, popisujícím celkovou situaci automatu. U konečných automatů byla konfigurace dána popisem situace na vstupní pásce (tj. specifikací dosud nepřečtené části vstupního slova) a vnitřní situací automatu (tj. specifikací jednoho, momentálního stavu, který reflektoval změny v automatu dané zpracováním již přečtené části vstupu počínaje počátečním stavem q_0). Co se týče vstupu, je situace u PDA identická, avšak zpracování již přečtené části vstupu se ve vnitřní situaci projeví nejen změnou stavu q_0, ale i tím, co si automat zapamatoval v zásobníku. Konfigurace by tedy měla opět popisovat jak vnitřní situaci jako dvojici (stav, obsah zásobníku), tak i dosud nepřečtenou část vstupního slova. Krok výpočtu pak bude definovat vztah mezi dvěma konfiguracemi – situací před provedením kroku výpočtu a situací po provedení tohoto kroku. Automat bude akceptovat vstupní slovo, jestliže bude existovat posloupnost kroků výpočtu začínající v jisté počáteční konfiguraci, která skončí v nějaké finální, akceptující konfiguraci.

Definice 3.37. Nechť $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je PDA. *Vnitřní konfigurací* (též totálním stavem) $\mathcal{M}$ nazveme libovolný prvek $(q, \gamma) \in Q \times \Gamma^*$ (kde q je momentální stav PDA $\mathcal{M}$ a γ je celý obsah zásobníku s vrcholem psaným vlevo). *Konfigurací* nazveme libovolný prvek (p, w, α) z $Q \times \Sigma^* \times \Gamma^*$ (udávající mimo totální stav navíc i w – dosud nepřečtenou

---

2. zápis $\mathcal{P}_{Fin}(Q \times \Gamma^*)$ značí množinu všech *konečných* podmnožin množiny $Q \times \Gamma^*$

část vstupního řetězu). Na množině všech konfigurací automatu $\mathcal{M}$ definujeme binární relaci $\vdash_{\mathcal{M}}$ (*krok výpočtu*) takto:

$$(p, aw, Z\alpha) \vdash_{\mathcal{M}} (q, w, \gamma\alpha) \overset{def}{\iff} \exists (q, \gamma) \in \delta(p, a, Z) \text{ pro } a \in \Sigma \cup \{\varepsilon\}$$

Reflexivní a tranzitivní uzávěr relace kroku výpočtu značíme $\vdash^{*}_{\mathcal{M}}$, její k-násobný součin značíme $\vdash^{k}_{\mathcal{M}}$. Je-li $\mathcal{M}$ zřejmý z kontextu, píšeme stručněji pouze $\vdash$, resp. $\vdash^{*}$, resp. $\vdash^{k}$.

Jazyk akceptovaný (též rozpoznávaný) PDA $\mathcal{M}$ *koncovým stavem* definujeme jako

$$L(\mathcal{M}) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^{*} (q_f, \varepsilon, \alpha), \text{ kde } q_f \in F, \alpha \in \Gamma^*\}$$

a jazyk akceptovaný (též rozpoznávaný) PDA $\mathcal{M}$ *prázdným zásobníkem* definujeme jako

$$L_e(\mathcal{M}) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^{*} (q, \varepsilon, \varepsilon), \text{ kde } q \in Q\}$$

Každý výpočet pro vstupní slovo w tedy začíná v konfiguraci (q_0, w, Z_0), tj. ve vnitřní konfiguraci (q_0, Z_0) a dosud nečteným vstupem. Způsobů akceptování je obecně více: každá akceptující (finální) konfigurace je charakterizována zcela přečteným vstupním slovem, tj. (q, ε, α), vzájemně se však tyto způsoby liší tím, co prohlásíme za akceptující totální stav (vnitřní konfiguraci). Ve výše uvedené definici 3.37 jsou to po řadě totální stavy z $F \times \Gamma^*$, resp. $Q \times \{\varepsilon\}$. Další způsoby akceptování lze definovat tak, že za akceptující vnitřní konfigurace prohlásíme např. prvky z $F \times \{\varepsilon\}$ (akceptování koncovým stavem a prázdným zásobníkem, resp. prvky z $Q \times \Gamma'\Gamma^*$ pro nějakou $\Gamma' \subset \Gamma$ (akceptování vrcholovými symboly v zásobníku). Formální definice jsou ponechány čtenáři.

Poznámka 3.38. *1. U takto definovaného PDA se opět jedná o nedeterministický automat, který akceptuje vstup, jestliže existuje alespoň jeden výpočet, který vede z konfigurace počáteční do konfigurace akceptující (při možnosti volby tedy PDA "hádá správně", protože nesprávná volba sama o sobě nemůže způsobit zamítnutí vstupu – ten může být zamítnut jedině tedy, pokud žádná správná volba neexistuje).*

2. Dále si povšimněme jedné důležité vlastnosti zásobníkových automatů, kterou lze parafrázovat takto: to, co se děje na vrcholu zásobníku (ve výše definovaném smyslu push-down store, *nikoli však obecnějším* stack*), děje se zcela nezávisle na tom, co je pod jeho vrcholem. Přesněji: pokud $(q, w, A) \vdash^{n} (q', \varepsilon, \varepsilon)$, pak $(q, w, A\alpha) \vdash^{n} (q', \varepsilon, \alpha)$ pro všechna $A \in \Gamma,\ \alpha \in \Gamma^*$. Toto tvrzení se snadno dokáže indukcí vzhledem k n.*

Věta 3.39. *Jazyk $L = L_e(\mathcal{M})$ pro nějaký PDA $\mathcal{M} \iff L = L(\mathcal{N})$ pro nějaký PDA $\mathcal{N}$.*

Idea důkazu: 1. $\Longleftarrow$: k danému $\mathcal{N}$ zkonstruujeme $\mathcal{M}$ simulující jeho činnost. Kdykoli $\mathcal{N}$ vejde do koncového stavu, bude $\mathcal{M}$ mít možnost volby, zda pokračovat v simulaci automatu $\mathcal{N}$ nebo přejít do svého, nově přidaného stavu q_e, v němž vyprázdní svůj zásobník. Musíme však uvážit možnou komplikaci: $\mathcal{N}$ může projít takovou posloupností kroků, kdy jeho vstup způsobí vymazání zásobníku a $\mathcal{N}$ není v koncovém stavu, tedy zamítá vstup. Musíme zabránit tomu, aby $\mathcal{M}$ v tomto bodě vstup (prázdným zásobníkem) akceptoval.

Řešení spočívá v tom, že ještě před zahájením simulace bude u $\mathcal{M}$ na dně zásobníku nový symbol, který nedovolíme odstranit jinde, než v koncovém stavu $q \in F$ nebo ve stavu q_e.

2. $\Longrightarrow$: $\mathcal{N}$ simuluje činnost $\mathcal{M}$ a má opět nově přidaný symbol jako své dno zásobníku. Jakmile je $\mathcal{N}$ schopen číst tento symbol (tj. zásobník simulovaného automatu $\mathcal{M}$ je prázdný), pak $\mathcal{N}$ přejde do nově přidaného stavu q_f, který je koncovým stavem.

Důkaz. 1. $\Longleftarrow$: Nechť $\mathcal{N} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je PDA takový, že $L = L(\mathcal{N})$. Sestrojme $\mathcal{M}$ takový, že $L = L_e(\mathcal{M})$. Položme

$$\mathcal{M} = (Q \cup \{q_0', q_e\}, \Sigma, \Gamma \cup \{Z'\}, \delta', q_0', Z', \emptyset) \ , \ \text{ kde } Z' \notin \Gamma \ \text{ a } q_0', q_e \notin Q$$

a kde δ' je definována takto:

1. $\delta'(q_0', \varepsilon, Z') = \{(q_0, Z_0 Z')\}$,
2. jestliže $\delta(q, a, Z)$ obsahuje (r, γ), pak $\delta'(q, a, Z)$ obsahuje (r, γ) pro všechna $q \in Q, a \in \Sigma \cup \{\varepsilon\}$ a $Z \in \Gamma$,
3. $\forall q \in F. \forall Z \in \Gamma \cup \{Z'\} : \ \delta'(q, \varepsilon, Z)$ obsahuje $(q_\varepsilon, \varepsilon)$,
4. $\forall Z \in \Gamma \cup \{Z'\} : \ \delta'(q_\varepsilon, \varepsilon, Z) = \{(q_\varepsilon, \varepsilon)\}$.

Pak zřejmě

$$\begin{aligned}(q_0', w, Z') &\underset{\mathcal{M}}{\vdash} (q_0, w, Z_0 Z') && -\text{dle } 1\\ &\underset{\mathcal{M}}{\overset{n}{\vdash}} (q, \varepsilon, Y_1 \dots Y_r) && -\text{dle } 2\\ &\underset{\mathcal{M}}{\vdash} (q_\varepsilon, \varepsilon, Y_2 \dots Y_r) && -\text{dle } 3\\ &\underset{\mathcal{M}}{\overset{r-1}{\vdash}} (q_\varepsilon, \varepsilon, \varepsilon) && -\text{dle } 4 \quad (\text{ kde } Y_r = Z')\end{aligned}$$

právě když $(q_0, w, Z_0) \underset{\mathcal{N}}{\overset{n}{\vdash}} (q, \varepsilon, Y_1 \dots Y_{r-1})$, pro nějaké $q \in F$.

2. $\Longrightarrow$: Nechť $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$ je PDA takový, že $L = L_e(\mathcal{M})$. Sestrojme $\mathcal{N}$ takový, že $L = L(\mathcal{N})$. Položme $\mathcal{N} = (Q \cup \{q_0', q_f\}, \Sigma, \Gamma \cup \{Z'\}, \delta', q_0', Z', \{q_f\})$ kde δ' je definována takto:

1. $\delta'(q_0', \varepsilon, Z') = \{(q_0, Z_0 Z')\}$
2. jestliže $\delta(q, a, Z)$ obsahuje (r, γ), pak $\delta'(q, a, Z)$ obsahuje (r, γ) pro všechna $q \in Q, a \in \Sigma \cup \{\varepsilon\}$ a $Z \in \Gamma$
3. $\forall q \in Q : \ \delta'(q, \varepsilon, Z') = \{(q_f, \varepsilon)\}$

Požadovaný důkaz, že $L_e(\mathcal{M}) = L(\mathcal{N})$, je obdobný jako v části 1 a je ponechán čtenáři.

□

Poznámka 3.40. *Ve výše uvedeném důkazu je prezentována technika zavedení nového symbolu Z' označující dno zásobníku simulujícího automatu, která umožnuje de facto testovat prázdnost zásobníku simulovaného automatu. Jestliže též požadujeme (srv. odstranění Z' pouze v q_ε), aby každá vnitřní konfigurace (s eventuelní výjimkou finální při akceptování prázdným zásobníkem) byla tvaru $(p, \gamma Z')$, budeme říkat, že takový PDA umožnuje* test dna zásobníku. *Pokud tedy PDA čte dno Z', musí jej též znovu na dno zapsat, tj. pokud δ obsahuje jako argument Z', pak je tvaru $\delta(p, a, Z') = \{(q_1, \gamma_1 Z'), \dots, (q_n, \gamma_n Z')\}$. Je zřejmé, že ke každému PDA $\mathcal{A}$ lze setrojit PDA $\mathcal{A}'$ s testem dna zásobníku akceptující tentýž jazyk.*

Poznamejme, že obdobně lze ukázat i ekvivalenci (ve smyslu výše uvedené věty 3.39) mezi akceptováním koncovým stavem a akceptováním koncovým stavem a (současně) prázdným zásobníkem či akceptováním vrcholovými symboly v zásobníku.

Příklad 3.41. *Mějme PDA* $\mathcal{P} = (\{p,q,r\},\{0,1\},\{Z,0\},\delta,p,Z,\{p\})$, *kde* δ *je definována takto:*

$$\begin{array}{lllllll} \delta(p,0,Z) & = & \{(q,0Z)\} & & & & \\ \delta(q,0,0) & = & \{(q,00)\} & & \delta(r,1,0) & = & \{(r,\varepsilon)\} \\ \delta(q,1,0) & = & \{(r,\varepsilon)\} & & \delta(r,\varepsilon,Z) & = & \{(p,\varepsilon)\} \end{array}$$

Automat $\mathcal{P}$ *pracuje tak, že nejprve ze vstupu načte do zásobníku všechny symboly '*0*' a následně s každým čteným (vstupním) symbolem '*1*' odstraní jeden symbol '*0*' z vrcholu zásobníku. Navíc se kontroluje, zda žádná '*1*' není před '*0*' (tím, že pro tyto situace je* δ *nedefinována, a tedy není definován krok výpočtu pro žádnou takovou situaci). Možná posloupnost (v obecném, nikoli však v tomto případě, pouze jedna z možných posloupností) konfigurací automatu* $\mathcal{P}$ *pro vstupní slovo například* 0011 *může být*

$$\begin{array}{lll} (p,0011,Z) & \vdash & (q,011,0Z) \\ & \vdash & (q,11,00Z) \\ & \vdash & (r,1,0Z) \\ & \vdash & (r,\varepsilon,Z) \\ & \vdash & (p,\varepsilon,\varepsilon) \end{array}$$

Lze ukázat, že $\mathcal{P}$ *akcetuje koncovým stavem jazyk* $L = \{0^n1^n|\ n \geq 0\}$. *Snadno se nahlédne, že* $L \subseteq L(\mathcal{P})$: *pro* $n \geq 1$ *lze po jediném přechodu z* p *do* q *(přečtení '*0*' a jeho uložení do zásobníku – viz 1. přechod výše) provést* n_1 *přechodů z* q *do* q *(opět čtení '*0*'a uložení do zásobníku – viz 2. přechod). Následně lze při čtení prvního symbolu '*1*' provést jediný přechod z* q *do* r *následovaný* n_1 *přechody z* r *do* r *po nichž zůstane v zásobníku pouze* Z*; následuje přechod do koncového stavu* p*. Případ* $n = 0$ *je triviální.*

Ukázat obrácenou inklusi (tj., že $\mathcal{P}$ *neakcetuje jiná slova než* 0^n1^n*) je obecně těžší úkol. Abychom to ukázali, uvědomme si, že při libovolném výpočtu nad neprázdným slovem musí automat procházet (s případnými cykly) postupně stavy* p,q,r *a skončí v* p*. Je-li* $(p,w,Z) \stackrel{i}{\vdash} (q,\varepsilon,\alpha), i \geq 1$, *pak* $w = 0^i$ *a* $\alpha = 0^iZ$*. Podobně, je-li* $(r,w,\alpha) \stackrel{i}{\vdash} (r,\varepsilon,\beta)$, $i \geq 1$, *pak* $w = 1^i$ *a* $\alpha = 0^i\beta$*. Dále přechod* $(q,w,\alpha) \vdash (r,\varepsilon,\beta)$ *nastane jedině tehdy, pokud* $w = 1$ *a* $\alpha = 0\beta$*; podobně* $(r,w,Z) \vdash (p,\varepsilon,\varepsilon)$, *pokud* $w = \varepsilon$*. Tedy celkem, je-li* $(p,w,Z) \stackrel{i}{\vdash} (p,\varepsilon,\alpha), i \geq 0$, *pak buď* $w = \varepsilon$ *a* $i = 0$ *nebo* $w = 0^i1^i, i = 2n+1$ *a* $\alpha = \varepsilon$*. Tedy* $L \supseteq L(\mathcal{P})$.

Zdůrazněme, že tak, jak byl PDA definován, může dělat (ε) kroky i po přečtení celého vstupu; PDA nemůže udělat žádný krok, pokud je zásobník prázdný.

Poznámka 3.42. O přechodových grafech ('stavových' diagramech) PDA. *Na stavové diagramy konečných automatů lze nahlížet tak, že jsme grafově znázornili vzájemné závislosti (dané přechodovou funkcí) mezi vnitřními konfiguracemi (tj. stavy) automatu. Podobně můžeme postupovat i v případě PDA* $\mathcal{P} = (N,\Sigma,\Gamma,\delta,q_0,Z,F)$*. Uzly* přechodového grafu *(stavového diagramu) budou označeny vnitřními konfiguracemi uvažovaného PDA (pro* *jednoduchost pišme vnitřní konfiguraci $\langle p, \alpha \rangle \in Q \times \Gamma^*$ ve tvaru $p\alpha \in Q\Gamma^*$). Analogicky jako u definice kroku výpočtu veďme z uzlu $pZ\alpha$ hranu s návěštím a ($a \in \Sigma \cup \{\varepsilon\}$) do uzlu $q\gamma\alpha$ právě když $\delta(p, a, Z)$ obsahuje (q, γ), což zapisujme jako $pZ\alpha \stackrel{a}{\to} q\gamma\alpha$.*

Poznamenejme, že stavový diagram PDA má obecně nekonečně mnoho uzlů (proto je PDA prvním příkladem "nekonečného" automatu). Část stavového diagramu pro $\mathcal{P}$ z příkladu 3.41 má tento tvar

$$\begin{tikzcd}
pZ \arrow[r, "0"] & q0Z \arrow[r, "0"] \arrow[d, "1"] & q00Z \arrow[r, "0"] \arrow[d, "1"] & q000Z \arrow[r, "0"] \arrow[d, "1"] & \ldots \arrow[r, "0"] & q0^iZ \arrow[r, "0"] \arrow[d, "1"] & \ldots \\
p\varepsilon & rZ \arrow[l, "\varepsilon"'] & r0Z \arrow[l, "1"'] & r00Z \arrow[l, "1"'] & \ldots \arrow[l, "1"'] & q0^{i-1}Z \arrow[l, "1"'] & \ldots \arrow[l, "1"']
\end{tikzcd}$$

kde koncovými totálními stavy jsou pZ a p, v diagramu pro názornost zapsaný jako $p\varepsilon$. Povšimněme si, že ve výše uvedeném stavovém diagramu automatu $\mathcal{P}$ jsou uvedeny pouze tzv. dosažitelné *totální stavy, tj. ty ke kterým existuje cesta z počátečního uzlu (zde pZ) končící v uzlu daném; ostatní nazveme nedosažitelné a obvykle je ve stavovém diagramu neuvádíme (ve výše uvedeném diagramu to jsou např. $qZZ, qZ0Z$ a další).*

V souladu s právě zavedenou notací lze při specifikaci funkce δ místo $\delta(p, a, Z) = \{(q_1, \gamma_1), \ldots, (q_n, \gamma_n)\}$ psát $pZ \stackrel{a}{\to} q_1\gamma_1 \mid \ldots \mid q_n\gamma_n$.

Relaci $\stackrel{a}{\to}$ můžeme zřejmým způsobem rozšířit ze symbolů ze Σ na řetězy nad touto abecedou ($\stackrel{aw}{\to} \stackrel{def}{=} \stackrel{a}{\to} \circ \stackrel{w}{\to}$); značení $\stackrel{w}{\to}, w \in \Sigma^*$ lze chápat jako zobecněnou přechodovou funkci pro PDA). V dalším textu budeme rovněž používat následující značení. Je-li $p\alpha$ nějaká vnitřní konfigurace PDA $\mathcal{P} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, pak definujme

$$L(\mathcal{P})(p\alpha) \stackrel{def}{=} \{w \in \Sigma^* \mid p\alpha \stackrel{w}{\to} q\beta, q \in F\} \quad \text{a} \quad L_e(\mathcal{P})(p\alpha) \stackrel{def}{=} \{w \in \Sigma^* \mid p\alpha \stackrel{w}{\to} q\varepsilon, q \in Q\}.$$

Zřejmě tedy $L(\mathcal{P}) = L(\mathcal{P})(q_0Z_0)$ a $L_e(\mathcal{P}) = L_e(\mathcal{P})(q_0Z_0)$. Bude-li $\mathcal{P}$ zřejmý z kontextu, budeme stručněji psát pouze $L(p\alpha)$ resp. $L_e(p\alpha)$.

Příklad 3.43. *Nechť $L = \{ww^R \mid w \in \{0,1\}^*\}$. Pak PDA rozpoznávající L prázdným zásobníkem je například $\mathcal{M} = (\{p, q\}, \{0, 1\}, \{R, G, B\}, \delta, p, R, \emptyset)$ (tj. množina koncových stavů nehraje žádnou roli), kde δ je definována (ve výše zavedené notaci) takto:*

(1) $pR \stackrel{0}{\to} pBR$	(4) $pG \stackrel{0}{\to} pBG$	(7) $qB \stackrel{0}{\to} q\varepsilon$
(2) $pR \stackrel{1}{\to} pGR$	(5) $pB \stackrel{1}{\to} pGB$	(8) $qG \stackrel{1}{\to} q\varepsilon$
(3) $pB \stackrel{0}{\to} pBB \mid q\varepsilon$	(6) $pG \stackrel{1}{\to} pGG \mid q\varepsilon$	(9) $pR \stackrel{\varepsilon}{\to} q\varepsilon$
		(10) $qR \stackrel{\varepsilon}{\to} q\varepsilon$

Pravidla (1) až (6) ukládají vstup na zásobník s tím, že pravidlech (3) a (6) je možnost alternativní volby: jestliže $\mathcal{M}$ uhodne, že bylo dosaženo středu vstupního slova, volí druhou alternativu; v ní pak přejde do stavu q, v němž postupně porovnává zbytek vstupního slova s obsahem zásobníku. Pokud $\mathcal{M}$ hádal správně a slovo bylo tvaru ww^R, pak dojde k přečtení celého slova a vyprázdnění zásobníku – vstupní slovo bylo akceptováno. Pokud by $\mathcal{M}$ hádal chybně a slovo bylo tvaru ww^R, pak by akceptující konfigurace nedosáhl, což ovšem neznamená zamítnutí vstupního slova – viz též pozn. 3.38.

Uvedený příklad ilustruje nedeterminismus PDA, avšak u PDA z př. 3.41 se možnosti volby způsobující nedeterminismus nevyskytly – v každé (vnitřní) konfiguraci je další krok určen jednoznačně.

Definice 3.44. *Rozšířeným* PDA nazveme $\mathcal{R} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde všechny symboly mají tentýž význam jako v definici PDA s výjimkou δ, která je zobrazením z *konečné podmnožiny* množiny $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma^*$ do konečných podmnožin množiny $Q \times \Gamma^*$. Pojmy konfigurace, kroku výpočtu, výpočtu a akceptovaného jazyka (koncovým stavem, prázdným zásobníkem) zůstávají rovněž beze změny.

Povšimněme si, že rozšířený PDA činí rozhodnutí o dalším kroku nikoli na základě jen jednoho (vrcholového) symbolu na zásobníku, ale na základě řetězu (konečné délky!), který je tvořen nejhornějšími symboly na zásobníku. Zápis $p\alpha \xrightarrow{a} q\beta$ (resp. $\delta(p, a, \alpha)$ obsahuje (q, β)) interpretujeme tak, že pokud má automat ve stavu p na vrcholu zásobníku α, pak po přečtení symbolu a ze vstupní pásky (analogicky pro ε-krok) vymaže ze zásobníku řetěz symbolů α, zapíše na něj řetěz symbolů β a změní svůj stav na q.

Specielně může rozšířený PDA učinit krok bez ohledu na zásobník (uvažuje řetěz ε), a tedy obecně je schopen dělat kroky i v situaci, kdy zásobník je prázdný.

Lemma 3.45. *Nechť $\mathcal{R}$ je rozšířený PDA. Pak existuje PDA $\mathcal{P}$ takový, že $L(\mathcal{P}) = L(\mathcal{R})$.*

Idea důkazu: Pro daný $\mathcal{R} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ označme m maximální délku řetězu symbolů na vrcholu zásobníku, který $\mathcal{R}$ používá k rozhodnutí o dalším kroku výpočtu (tj. $m = \max\{|\alpha|;\ \delta(q, a, \alpha) \neq \emptyset,$ pro nějaká $q \in Q, a \in \Sigma \cup \{\varepsilon\}\}$).

Budeme simulovat $\mathcal{R}$ tak, že k rozhodování o dalším kroku potřebných m symbolů, které má $\mathcal{R}$ k dispozici na zásobníku, si bude simulující PDA $\mathcal{P}$ uchovávat (a aktualizovat) ve "vyrovnávací paměti" (konečné) délky m (tu bude mít ve své konečně stavové řídící jednotce – stavy automatu $\mathcal{P}$ budou tedy dvojice ¡stav simulovaného $\mathcal{R}$, stav vyrovnávací paměti¿). Tedy $\mathcal{P}$ bude vědět před provedením každého kroku, kterých m symbolů má $\mathcal{R}$ na vrcholu zásobníku. Zmíněná aktualizace probíhá takto: je-li v $\mathcal{R}$ nahrazeno k vrcholových symbolů, řekněme α, l symboly, řekněme β, pak v paměti PDA $\mathcal{P}$ nahradíme též těchto k symbolů týmiž l symboly; Je-li $l < k$, pak $\mathcal{P}$ udělá navíc $k - l$ aktualizačních kroků, při nichž postupně přenáší symboly z vrcholu zásobníku do paměti. Je-li $l > k$, je zapotřebí (ještě před náhradou α za β) "nadbytečných" $l - k$ symbolů z konečné vyrovnávací paměti postupně přesunout na zásobník. V obou případech tak bude vyrovnávací paměť automatu $\mathcal{P}$ opět obsahovat přesně m symbolů, které má $\mathcal{R}$ momentálně na vrcholu na zásobníku.

Důkaz. Nechť $\mathcal{R} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je rozšířený PDA a označme $m = \max\{|\alpha|;\ \delta(q, a, \alpha) \neq \emptyset,$ pro nějaká $q \in Q, a \in \Sigma \cup \{\varepsilon\}\}$. Nyní definujme PDA $\mathcal{P} = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, Z_1, F_1)$, kde

1. $Q_1 = \{[q, \alpha] \mid q \in Q, \alpha \in \Gamma_1^*, 0 \leq |\alpha| \leq m\}$,
2. $\Gamma_1 = \Gamma \cup \{Z_1\}$, kde Z_1 je nový symbol,
3. δ_1 je definována takto:
   (a) Nechť $\delta(q, a, X_1 \dots X_k)$ obsahuje $(r, Y_1 \dots Y_l)$. Pak
      i. je-li $l \geq k$, pak pro všechna $Z \in \Gamma_1$ a $\alpha \in \Gamma_1^*$ taková, že $|\alpha| = m - k$, klademe $\delta_1([q, X_1 \dots X_k\alpha], a, Z)$ obsahuje $([r, \beta], \gamma Z)$, kde $\beta\gamma = Y_1 \dots Y_l\alpha$ a $|\beta| = m$;
      ii. je-li $l < k$, pak pro všechna $Z \in \Gamma_1$ a $\alpha \in \Gamma_1^*$ taková, že $|\alpha| = m - k$,

klademe $\delta_1([q, X_1 \dots X_k\alpha], a, Z)$ obsahuje $([r, Y_1 \dots Y_l\alpha Z], \varepsilon)$.

(b) pro všechna $q \in Q, Z \in \Gamma_1$ a $\alpha \in \Gamma_1^*$ taková, že $|\alpha| < m$, klademe $\delta_1([q, \alpha], \varepsilon, Z) = \{([q, \alpha Z], \varepsilon)\}$

4. $q_1 = [q_0, Z_0 Z_1^{m-1}]$
5. $F_1 = \{[q, \alpha] \mid q \in F, \alpha \in \Gamma_1^*\}$.

Na základě takto definované δ_1 není obtížné ověřit, že

$$(q, aw, X_1 \dots X_k X_{k+1} \dots X_n) \vdash_{\mathcal{R}} (r, w, Y_1 \dots Y_l X_{k+1} \dots X_n)$$

$$\iff ([q, \alpha], aw, \beta) \vdash_{\mathcal{P}}^{+} ([r, \alpha'], w, \beta'), \text{ kde}$$

1. $\alpha\beta = X_1 \dots X_n Z_1^m$,
2. $\alpha'\beta' = Y_1 \dots Y_l X_{k+1} \dots X_n Z_1^m$,
3. $|\alpha| = |\alpha'| = m$ a
4. mezi dvěma výše uvedenými konfiguracemi PDA $\mathcal{P}$ neexistuje taková konfigurace, kde druhá komponenta stavu (tj. paměť) by měla délku m.

Odtud pak okamžitě plyne, že $(q_0, w, Z_0) \vdash_{\mathcal{R}}^{*} (q, \varepsilon, \alpha)$ pro nějaká $q \in F$ a $\alpha \in \Gamma^*$ právě když $([q_0, Z_0 Z_1^{m-1}], w, Z_1) \vdash_{\mathcal{P}}^{*} ([q, \beta], \varepsilon, \gamma)$, kde $|\beta| = m$ a $\beta\gamma = \alpha Z_1^m$. Tedy $L(\mathcal{R}) = L(\mathcal{P})$. □

Poznámka 3.46. *Poznamenejme, že i pro rozšířené PDA platí tvrzení věty 3.39 o ekvivalenci rozpoznávání prázdným zásobníkem a koncovým stavem. Toto tvrzení se dokáže naprosto stejně jako u zmíněné věty.*

3.2.2 Zásobníkové automaty a bezkontextové jazyky

V této části ukážeme fundamentální výsledek, že třída jazyků rozpoznávaných zásobníkovými automaty tvoří právě třídu bezkontextových jazyků.

Věta 3.47. (O nedeterministické syntaktické analýze shora dolů) *Nechť $\mathcal{G}$ je libovolná CFG. Pak lze sestrojit PDA $\mathcal{M}$ takový, že $L(\mathcal{G}) = L_e(\mathcal{M})$.*

Idea důkazu: Ke $\mathcal{G}$ zkonstruujeme (jednostavový) PDA tak, aby ve svém zásobníku byl schopen simulovat levé derivace v $\mathcal{G}$, přičemž budeme požadovat, aby v každé konfiguraci platilo:

$$\left.\begin{array}{l} \mathcal{M} \text{ z konfigurace s obsahem zásobníku } \alpha \text{ a zbytkem vstupu } w \\ \text{akceptuje prázdným zásobníkem} \iff \text{v } \mathcal{G} \text{ lze derivovat } \alpha \Rightarrow^* w \end{array}\right\} \quad (*)$$

V $\mathcal{G}$ je v jednom kroku odvození nahrazen (nejlevější) neterminál A (naráz, celou) pravou stranou $X_1 \dots X_n$, kdežto v $\mathcal{M}$ bude této situaci odpovídat (1): náhrada neterminálu A na vrcholu zásobníku toutéž pravou stranou následovaná (2): postupným (symbol po symbolu) zpracováním této, na vrchol zásobníku přidané, pravé strany: nechť se má zpracovat (tj. je na vrcholu zásobníku) X_i. Je-li X_i neterminál, postup ad (1) opakujeme, je-li (2) X_i terminál, pak zkontrolujeme (tj. ověřuje se korektnost nedeterministické volby v kroku typu (1)) zda X_i je stejný jako první, dosud nečtený terminál na vstupu; pokud ano, pak terminál z vrcholu zásobníku odstraníme a čtecí hlava se posune o jeden symbol vpravo.

Korektnost uvedeného postupu spočívá v důkazu, že v jeho průběhu platí tvrzení (*). Odstartujeme-li $\mathcal{M}$ v situaci, kdy v zásobníku je pouze kořen (a na vstupu slovo z $L(\mathcal{G})$), pak (*) platí. Dále se (indukcí) ověří, že operace (1) a (2) zachovávají platnost (*).

Pro pochopení činnosti $\mathcal{M}$ je vhodné si uvědomit, že (*) implikuje tuto (opět v celém procesu invariantní) vlastnost:

$$\left.\begin{array}{l}\text{dosud přečtená část vstupu zřetězena s obsahem zásobníku v } \mathcal{M} \\ \text{je odpovídající levou větnou formou v } \mathcal{G} \text{ (a obráceně)}\end{array}\right\} \qquad (**)$$

(tedy vstupní slovo je z $L(\mathcal{G})$, právě když celý proces skončí v akceptující konfiguraci s celým přečteným vstupem a prázdným zásobníkem).

Důkaz. Nechť $\mathcal{G} = (N, \Sigma, P, S)$. Definujme $\mathcal{M}$ akceptující prázdným zásobníkem jako

$\mathcal{M} = (\{q\}, \Sigma, N \cup \Sigma, \delta, q, S, \emptyset)$, kde δ je definována takto:

(1) $qA \stackrel{\varepsilon}{\to} q\alpha \in \delta$, je-li $A \to \alpha \in P$ (tj. $\delta(q, \varepsilon, A)$ obs. (q, α))

(2) $qa \stackrel{a}{\to} q\varepsilon \in \delta$, pro všechna $a \in \Sigma$ (tj. $\delta(q, a, a) = \{(q, \varepsilon)\}$)

Abychom ukázali $L(\mathcal{G}) = L_e(\mathcal{M})$, ukažme, že pro nějaká $m, n \geq 1$ platí:

$$A \Rightarrow^m w \iff (q, w, A) \vdash^{n} (q, \varepsilon, \varepsilon) \quad \text{(tj. } qA \stackrel{w}{\to} q\varepsilon \text{ v } n \text{ krocích)} \qquad (*1)$$

1. $\Longrightarrow$: Mějme $A \Rightarrow^m w$ a ukažme, že $(q, w, A) \vdash^{*} (q, \varepsilon, \varepsilon)$ (tj. $qA \stackrel{w}{\to} q\varepsilon$):

(a) je-li $m = 1$ a $w = a_1 \dots a_k$, $(k \geq 0)$, pak zřejmě $A \to w \in P$, a tedy

$(q, a_1 \dots a_k, A) \vdash (q, a_1 \dots a_k, a_1 \dots a_k) \vdash^{k} (q, \varepsilon, \varepsilon)$

(b) Přepokládejme, že (*1) platí pro všechna $m' < m$ a nechť $A \Rightarrow^m w$ pro $m > 1$. Pak 1. krok derivace je tvaru $A \Rightarrow X_1 X_2 \dots X_k$, kde $X_i \Rightarrow^{m_i} x_i, 0 \leq m_i < m$, což dle definice δ, bodu (1) dává $(q, w, A) \vdash (q, w, X_1 X_2 \dots X_k)$. (1.1)

Je-li $X_i \in N$, pak dle indukčního předpokladu máme $(q, x_i, X_i) \vdash^{*} (q, \varepsilon, \varepsilon)$. (1.2)

Je-li $X_i \in \Sigma$ (tj. $X_i = x_i$), pak dle definice δ, bodu (2) máme $(q, x_i, x_i) \vdash (q, \varepsilon, \varepsilon)$. (1.3)

Kompozicí přechodů (1.1) – (1.3) tedy dostáváme $(q, w, A) \vdash^{+} (q, \varepsilon, \varepsilon)$.

2. $\Longleftarrow$: Předpokládejme, že $(q, w, A) \vdash^{n} (q, \varepsilon, \varepsilon)$ a ukažme, že $A \Rightarrow^+ w$:

(a) $n = 1$ implikuje $qA \stackrel{\varepsilon}{\to} q\varepsilon \in \delta$, a tedy $w = \varepsilon$ a $A \to \varepsilon \in P$.

(b) Předpokládejme, že dokazované tvrzení platí pro všechna $n' < n$. Pak 1. krok je tvaru

$(q, w, A) \vdash (q, w, X_1 X_2 \dots X_k)$, tj. $A \to X_1 X_2 \dots X_k \in P$ (2.1)

Dále $(q, x_i, X_i) \vdash^{n_i} (q, \varepsilon, \varepsilon)$, kde $n_i < n, 1 \leq i \leq k$ a kde $w = x_1 x_2 \dots x_k$ je takové, že

je-li $X_i \in N$, pak dle indukčního předpokladu máme $X_i \Rightarrow^+ x_i$, (2.2)

je-li $X_i \in \Sigma$, pak $X_i \Rightarrow^0 x_i$. (2.3)

Vhodnou kompozicí (2.1) – (2.3) tedy obdržíme

$$\begin{aligned} A &\Rightarrow X_1 X_2 \dots X_k \\ &\Rightarrow^* x_1 X_2 \dots X_k \\ &\vdots \\ &\Rightarrow^* x_1 \dots x_k = w \end{aligned}$$

, což je (korektní) levá derivace slova w v gramatice $\mathcal{G}$.

Položíme-li $A = S$ v právě dokázaném tvrzení (*1), okamžitě dostáváme žádané $S \Rightarrow^+ w \iff (q, w, S) \vdash^{+} (q, \varepsilon, \varepsilon)$, tj. $qS \stackrel{w}{\to} q\varepsilon$. □

Poznámka 3.48. *Poznamenejme, že uvedený důkaz lze modifikovat tak, že (bez újmy na* *obecnosti) předpokládáme, že daná CFG je v GNF. Pak body (1) a (2) v definici přechodové funkce δ lze nahradit jediným, a to $qA \overset{a}{\to} q\alpha \in \delta$, je-li $A \to a\alpha$. Princip důkazu ovšem zůstává beze změny. (Přechodový graf takto vzniklého PDA k dané CFG $\mathcal{G}$, resp. k ekvivalentní CFG v GNF, nazveme rovněž* přechodovým grafem CFG $\mathcal{G}$.*) Prezentovaná varianta byla zvolena proto, že (dle našeho názoru) zřetelněji artikuluje nedetermininismus vznikající právě v bodu (1) definice δ (viz též poznámka 3.50) a nevyžaduje převod do GNF.*

Příklad 3.49. *Mějme gramatiku $\mathcal{G}_0 = (\{E, T, F\}, \{+, *, (,), i\}, P, E)$ s pravidly P danými takto:*

$$E \to E{+}T \mid T$$
$$T \to T{*}F \mid F$$
$$F \to (E) \mid i$$

Pak PDA sestrojený dle konstrukce z důkazu věty 3.47 je

$\mathcal{P} = (\{q\}, \{+, *, (,), i\}, \{E, T, F, +, *, (,), i\}, \delta, q, E, \emptyset)$, *kde δ je definována takto:*

$qE \overset{\varepsilon}{\to} qE{+}T \mid qT$	*dle bodu (1)*
$qT \overset{\varepsilon}{\to} qT{*}F \mid qF$	*dle bodu (1)*
$qF \overset{\varepsilon}{\to} q(E) \mid qi$	*dle bodu (1)*
$qa \overset{a}{\to} q\varepsilon$ pro všechna $a \in \{+, *, (,), i\}$	*dle bodu (2)*

Akceptující výpočet automatu $\mathcal{P}$ pro vstupní slovo $i{+}i{*}i$ je uveden na obrázku 3.3. Jelikož $\mathcal{P}$ je jednostavový, nebudeme stav v konfiguracích uvádět; tyto jsou tedy dvojicemi tvaru (vstup, obsah zásobníku). Doporučujeme ověřit si platnost vlastnosti (*) a (**) spefikované v idei důkazu věty 3.47.

krok	odpovídající pravidlo z $\mathcal{G}_0$
$(i+i*i,\ E) \vdash^{\varepsilon} (i+i*i,\ E+T)$	$E \to E+T$
$\vdash^{\varepsilon} (i+i*i,\ T+T)$	$E \to T$
$\vdash^{\varepsilon} (i+i*i,\ F+T)$	$T \to F$
$\vdash^{\varepsilon} (i+i*i,\ i+T)$	$F \to i$
$\vdash^{i} (+i*i,\ +T)$	
$\vdash^{+} (i*i,\ T)$	
$\vdash^{\varepsilon} (i*i,\ T*F)$	$T \to T*F$
$\vdash^{\varepsilon} (i*i,\ F*F)$	$T \to F$
$\vdash^{\varepsilon} (i*i,\ i*F)$	$F \to i$
$\vdash^{i} (*i,\ *F)$	
$\vdash^{*} (i,\ F)$	
$\vdash^{\varepsilon} (i,\ i)$	$F \to i$
$\vdash^{i} (\ ,\ )$	

Obrázek 3.3: Výpočet automatu $\mathcal{P}$ z příkladu 3.49 pro vstupní slovo $i + i * i$

Poznámka 3.50. *Jestliže PDA nejen koretně rozhoduje zda $w \in L(\mathcal{G})$, ale při kladné odpovědi je navíc schopen určit derivační strom vstupní věty, říkáme, že PDA provádí* syntaktickou analýzu. *Ve výše uvedeném příkladu 3.49 je posloupnost odpovídajících pravidel* *z $\mathcal{G}_0$ použitých při výpočtu v $\mathcal{P}$ posloupností pravidel použitých při levé derivaci vstupní věty. Pokud bychom na základě této posloupnosti postupně konstruovali odpovídající derivační strom, povšimneme si, že konstrukce začíná u jeho kořene a jde směrem k listům; odtud název této techniky – nedeterministická syntaktická analýza shora dolů. Nedeterminismus se objevuje jen v konfiguracích, kdy PDA má na vrcholu neterminál a volí, kterou z odpovídajích pravých stran jej nahradit.*

Věta 3.51. *Ke každému PDA $\mathcal{M}$ lze sestrojit CFG $\mathcal{G}$ takovou, že $L_e(\mathcal{M}) = L(\mathcal{G})$.*

Idea důkazu: Nejprve si uvědomme, že pokud by PDA byl jednostavový, pak nalezení ekvivaletní CFG by bylo velmi snadné – konstrukce z důkazu věty 3.47 resp. poznámky 3.48 (tj. k CFG sestrojit ekvivalentní PDA) je plně reversibilní. V podstatě tedy půjde o tento cíl: (***)
k danému PDA $\mathcal{M}$ nalézt ekvivaletní PDA $\mathcal{M}'$ s jedním stavem (označeným např. symbolem '*') takový, že levé odvození v hledané $\mathcal{G}$ má být simulací práce $\mathcal{M}'$ (tj. neterminály, které se vyskytují v libovolném kroku levé derivace v $\mathcal{G}$, odpovídají symbolům v zásobníku automatu $\mathcal{M}'$ v době, kdy tento již přečetl ze vstupu to, co $\mathcal{G}$ nalevo od nejlevějšího neterminálu vygenerovala (srovnej s vlastnosti (*) resp. (**) uvedenými v idei důkazu zmíněné věty).

Klíčovým úkolem důkazu je tedy ukázat, jak simulovat libovolný PDA $\mathcal{M}$ jednostavovým PDA $\mathcal{M}'$ – oba dva akceptující prázdným zásobníkem. Informaci o stavu simulovaného PDA nelze v simulujícím jednostavovém PDA umístit jinam než na zásobník, tj. "vhodným způsobem" ji na zásobník přidat.

Je-li $w \in L_e(\mathcal{M})$, pak $q_0 Z_0 \xrightarrow{w} q$ do nějakého $q \in Q$, jinak řečeno $\mathcal{M}$ odstartován v $q_0 Z_0$ přečetl w a skončil vymazáním zásobníku v nějakém q. Pak v souladu s naším cílem (***) je přirozené požadovat dostupnost přesně této informace v zásobníku jednostavového PDA: požadujme po $\mathcal{M}'$, aby odstartován s informací tvaru $\langle q_0 Z_0 q\rangle$ jako jediným symbolem v zásobníku byl schopen přečíst vstup w a akceptovat jej prázdným zásobníkem, tj. mít v hledané gramatice neterminál tvaru $\langle q_0 Z_0 q\rangle$ takový, aby $\langle q_0 Z_0 q\rangle \Rightarrow^* w$. Uvědomme si, že v průběhu celého výpočtu zásobník neklesl – s výjimkou poslední konfigurace – pod hladinu danou vrcholovým symbolem Z_0 v počátečním totálním stavu $q_0 Z_0$.

Původní PDA $\mathcal{M}$ ovšem prochází při svém (výše naznačeném) výpočtu celou posloupností kroků, a proto je přirozené zjemnit informaci $q_0 Z_0 \xrightarrow{w} q$ (resp. $\langle q_0 Z_0 q\rangle \Rightarrow^* w$) tak, že jednostavový PDA $\mathcal{M}'$ bude mít v zásobníku symboly tvaru $\langle pAq\rangle$ z $Q\Gamma Q$ takové, aby platilo: $\mathcal{M}'$ odstartován s $\langle pAq\rangle$ jako jediným symbolem ve svém zásobníku akceptuje vstup x prázdným zásobníkem právě když $\mathcal{M}$ odstartován ve vnitřní konfiguraci pA akceptuje x prázdným zásobníkem ve stavu q, tj. (připomeňme, že symbol '*' reprezentuje jediný stav PDA $\mathcal{M}'$):

$$*\langle pAq\rangle \xrightarrow{x} *\varepsilon \ \text{v}\ \mathcal{M}' \iff pA \xrightarrow{x} q \ \text{v}\ \mathcal{M}\ .$$

Nyní zbývá definovat přechody v $\mathcal{M}'$. S každým přechodem

$$pA \xrightarrow{a} q_1 B_1 \dots B_n \ (a \in \Sigma \cup \{\varepsilon\}) \ \text{v}\ \mathcal{M}$$

přidejme do $\mathcal{M}'$ všechny přechody tvaru

$$*\langle pAq\rangle \xrightarrow{a} *\langle q_1 B_1 q_2\rangle\langle q_2 B_2 q_3\rangle \dots \langle q_n B_n q_{n+1}\rangle$$

pro všechny možné volby $q_2, \dots, q_{n+1}$ takové, že $q_{n+1} = q$. V případě $n = 0$, tedy

$pA \xrightarrow{a} q\varepsilon$ v $\mathcal{M}$, přidáme do $\mathcal{M}'$ přechod $*\langle pAq\rangle \xrightarrow{a} *\varepsilon$.

Intuitivně řečeno, $\mathcal{M}'$ simuluje $\mathcal{M}$ tak, že nedeterministicky hádá, v kterých stavech se $\mathcal{M}$ ve svém budoucím výpočtu ocitne: uloží si tato hádání na zásobník s tím, že posléze kontroluje korektnost svého nedeterministického hádání.

V následujícím důkazu explicitní konstrukci $\mathcal{M}'$ vypustíme a budeme pracovat přímo s hledanou $\mathcal{G}$, protože je zřejmé, že každému přechodu $*\langle pAq\rangle \xrightarrow{a} *\langle q_1B_1q_2\rangle\langle q_2B_2q_3\rangle\dots$ $\langle q_nB_nq_{n+1}\rangle$ v $\mathcal{M}'$ odpovídá v gramatice přímo pravidlo $\langle pAq\rangle \to a\langle q_1B_1q_2\rangle\langle q_2B_2q_3\rangle\dots$ $\langle q_nB_nq_{n+1}\rangle$. Navíc, protože gramatika může mít jen jeden kořen (resp. PDA jen jeden počáteční symbol v zásobníku), kdežto symbolů tvaru $\langle q_0Z_0q\rangle$ je obecně více, přidáme i pravidla $S \to \langle q_0Z_0q\rangle$ pro všechna $q\in Q$ (pro $\mathcal{M}$ platí $L_e(\mathcal{M}) = \{w \mid q_0Z_0 \xrightarrow{w} q, q\in Q\}$).

Důkaz. Nechť PDA $\mathcal{M}=(Q,\Sigma,\Gamma,\delta,q_0,Z_0,\emptyset)$ a nechť $\mathcal{G}=(N,\Sigma,P,S)$ je CFG, kde

- $N = \{S\} \cup \{\langle pAq\rangle \mid p,q \in Q, A \in \Gamma$, kde $S \notin Q\cup\Sigma\cup\Gamma$ je nový symbol
- P obsahuje pravidla:
  1. $S \to \langle q_0Z_0q\rangle$ pro všechna $q \in Q$
  2. $\langle pAq\rangle \to a\langle q_1B_1q_2\rangle\langle q_2B_2q_3\rangle\dots\langle q_nB_nq_{n+1}\rangle$ pro $q_{n+1}=q, a\in\Sigma\cup\{\varepsilon\}$, $A, B_i(1\le i\le n) \in \Gamma, q_i \in Q$, je-li $pA \xrightarrow{a} q_1B_1B_2\dots B_n \in \delta$;

     je-li $n=0$, tj. $pA \xrightarrow{a} q_1$, pak klademe $q_1 = q$ a $\langle pAq\rangle \to a \in P$.

Abychom ukázali, že $L(\mathcal{G}) = L_e(\mathcal{M})$, tj. $S \Rightarrow \langle q_0Z_0q\rangle \Rightarrow^* w \iff q_0Z_0 \xrightarrow{w} q\varepsilon$, dokažme:

$$\langle pAq\rangle \Rightarrow^* x \iff pA \xrightarrow{x} q\varepsilon \tag{1}$$

1. $\Longleftarrow$: Dokazujme, že $(p,x,A) \overset{i}{\vdash} (q,\varepsilon,\varepsilon)$ implikuje $\langle pAq\rangle \Rightarrow^* x$, a to indukcí vzhledem k i (vzhledem k čemu ještě by šlo indukci vést?).

(a) Je-li $i=1$, pak jistě $pA \xrightarrow{a} q\varepsilon \in \delta$ $(a\in\Sigma\cup\{\varepsilon\})$, a tedy $\langle pAq\rangle \to a$ je pravidlo z P.

(b) Nechť $i>1$ a nechť $x = ay$ a

$$(p,ay,A) \vdash (q_1,y,B_1B_2\dots B_n) \overset{i-1}{\vdash} (q,\varepsilon,\varepsilon). \tag{1.1}$$

Řetěz y lze zapsat ve tvaru $y = y_1y_2\dots y_n$ takovém, že přečtení y_i způsobí odstranění B_i ze zásobníku (posloupností kroků, kdy zásobník může ještě vzrůst, ale nikdy neklesne pod úroveň, na níž je B_i). Jinak řečeno, y_1 je takovou předponou slova y, že po jejím přečtení se zásobník poprvé zkrátí na úroveň (hloubku) $n-1$ (bude obsahovat $B_2\dots B_n$); y_2 je takový řetěz, že následuje za y_1 a po přečtení y_2 se poprvé zásobník zkrátí na úroveň $n-2$ atd. Podstatné je, že během zpracování y_i se žádné z $B_{i+1}\dots B_n$ neobjevilo na vrcholu zásobníku, a tudíž nemohlo ovlivnit průběh této části výpočtu, tj. B_j zůstává na zásobníku bez vlivu na výpočet nad $y_1\dots y_{j-1}$ – viz obrázek 3.4.

Tedy existují stavy $q_2,q_3,\dots,q_{n+1}$ (kde $q_{n+1}=q$) takové, že výpočet

$$(q_j,y_j,B_j) \overset{*}{\vdash} (q_{j+1},\varepsilon,\varepsilon)$$

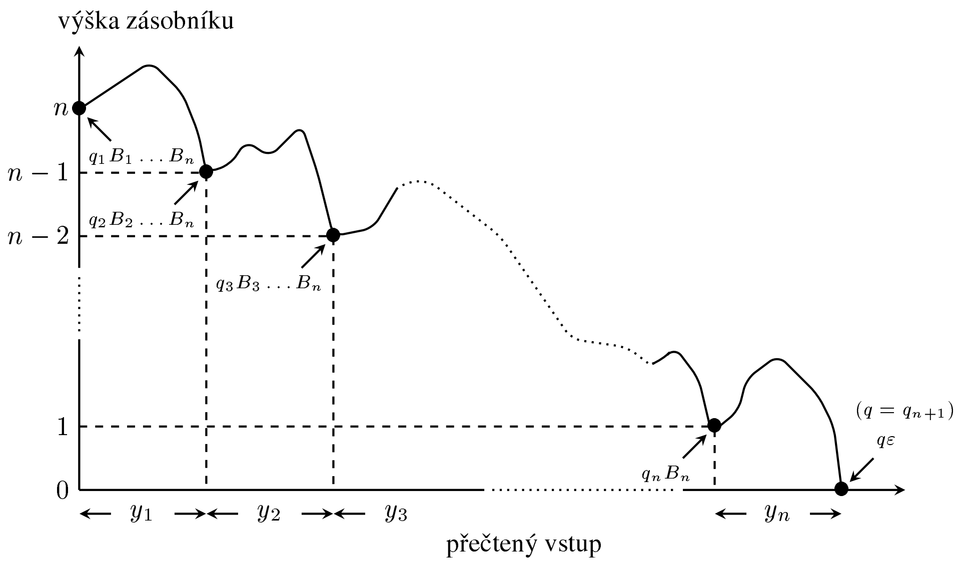
proběhne v méně než i krocích (q_j je stav, do kterého se automat dostal, když se zásobník poprvé zkrátil na úroveň $n-j+1$). Podle indukčního předpokladu tedy dostáváme

$$\langle q_jB_jq_{j+1}\rangle \Rightarrow^* y_j \quad \text{pro všechna } 1\le j\le n \tag{1.2}$$

Protože první krok celého výpočtu (1.1) nad $x = ay$ byl $(p,ay,A) \vdash (q_1,y,B_1B_2\dots B_n)$ (tj. $pA \xrightarrow{a} q_1B_1B_2\dots B_n \in \delta$), máme též

$$\langle pAq\rangle \Rightarrow a\langle q_1B_1q_2\rangle\langle q_2B_2q_3\rangle\dots\langle q_nB_nq_{n+1}\rangle,$$

což spolu s (1.2) dává žádané $\langle pAq\rangle \Rightarrow^* ay_1y_2\dots y_n = x$.



Obrázek 3.4: Výška zásobníku jako funkce přečteného vstupu – k důkazu Věty 3.51

2. $\Longrightarrow$: Nyní předpokládejme, že $\langle pAq \rangle \Rightarrow^i x$ a indukcí vzhledem k i ukažme, že pak $(p, x, A) \vdash^* (q, \varepsilon, \varepsilon)$.

(a) Je-li $i = 1$, pak $\langle pAq \rangle \to x$ je pravidlo v $\mathcal{G}$, což značí, že $pA \xrightarrow{x} q \in \delta$ $(x \in \Sigma \cup \{\varepsilon\})$.
(b) Nechť $i > 1$. Derivaci $\langle pAq \rangle \Rightarrow^i x$ lze zapsat jako

$$\langle pAq \rangle \Rightarrow a\langle q_1B_1q_2\rangle\langle q_2B_2q_3\rangle \dots \langle q_nB_nq_{n+1}\rangle \Rightarrow^{i-1} x, \quad \text{kde } q_{n+1} = q \qquad (2.1).$$

Pak x lze zapsat ve tvaru $x = ax_1x_2 \dots x_n$ takovém, že pro každé $j(1 \le j \le n)$ platí $\langle q_jB_jq_{j+1}\rangle \Rightarrow^* x_j$, kde každá tato derivace má méně než i kroků. Tedy dle indukčního předpokladu platí $(q_j, x_j, B_j) \vdash^* (q_{j+1}, \varepsilon, \varepsilon)$ pro všechna $j(1 \le j \le n)$. Pokud v každé této posloupnosti konfigurací vložíme na dno zásobníku řetěz $B_{j+1} \dots B_n$, obdržíme

$$(q_j, x_j, B_jB_{j+1} \dots B_n) \vdash^* (q_{j+1}, \varepsilon, B_{j+1} \dots B_n). \qquad (2.2)$$

Z prvního kroku derivace (2.1) máme, že $(pA \xrightarrow{a} q_1B_1B_2 \dots B_n) \in \delta$, tj.

$$(p, x, A) \vdash (q_1, x_1x_2 \dots x_n, B_1B_2 \dots B_n)$$

je korektní krok, což spolu s (2.2) pro $j = 1, 2, \dots, n$ dává žádané $(p, x, A) \vdash^* (q, \varepsilon, \varepsilon)$.

Jestliže v právě dokázaném tvrzení (1) položíme $p = q_0$ a $A = Z_0$, dostáváme

$$\langle q_0Z_0q \rangle \Rightarrow^* x \iff q_0Z_0 \xrightarrow{x} q\varepsilon,$$

což spolu s pravidlem 1. pro konstrukci množiny P pravidel gramatiky $\mathcal{G}$ dává

$$S \Rightarrow^* x \iff q_0Z_0 \xrightarrow{x} q\varepsilon \text{ pro nějaký stav } q \in Q,$$

tj. $x \in L(\mathcal{G}) \iff x \in L_e(\mathcal{M})$, což jsme měli dokázat. □

Sumarizujme dosud dosažené výsledky o rozpoznávání CFL pomocí PDA.

Důsledek 3.52. *1. $L = L(\mathcal{G})$ pro nějakou CFG $\mathcal{G}$*
2. $L = L(\mathcal{M})$ pro nějaký PDA $\mathcal{M}$
3. $L = L_e(\mathcal{N})$ pro nějaký PDA $\mathcal{N}$

4. $L = L(\mathcal{P})$ *pro nějaký rozšířený PDA* $\mathcal{P}$.

Důkaz. $3 \Rightarrow 1$ (dle 3.47); $1 \Rightarrow 3$ (3.47); $4 \Rightarrow 2$ (3.45); $2 \Rightarrow 4$ (triviální); $2 \Longleftrightarrow 3$ (3.39). □

Příklad 3.53. *Mějme dán PDA* $\mathcal{M} = (\{q_0, q_1\}, \{a, b\}, \{X, Z_0\}, \delta, q_0, Z_0, \emptyset)$, *kde* δ *je definována takto:*

$$q_0 Z_0 \xrightarrow{a} q_0 X Z_0 \qquad q_1 X \xrightarrow{b} q_1$$
$$q_0 X \xrightarrow{a} q_0 X X \qquad q_1 X \xrightarrow{\varepsilon} q_1$$
$$q_0 X \xrightarrow{b} q_1 \qquad q_1 Z_0 \xrightarrow{\varepsilon} q_1$$

a konstruujme CFG $\mathcal{G} = (N, \Sigma, P, S)$ *tak, aby generovala* $L_e(\mathcal{M})$.

Zřejmě $\Sigma = \{a, b\}$ a $N = \{S\} \cup \{\langle q_i X q_j\rangle | i, j \in \langle 0, 1\rangle\} \cup \{\langle q_i Z_0 q_j\rangle | i, j \in \langle 0, 1\rangle\}$.

Při konstrukci množiny pravidel P *lze postupovat systematicky tak, že začneme s kořenovými* S*-pravidly a další pravidla přidáváme jen pro ty neterminály* $\langle \ldots \rangle$, *které se již vyskytly na některé pravé straně do* P *již přidaného pravidla (proč?). Tedy* S*-pravidla jsou*

$$S \to \langle q_0 Z_0 q_0\rangle \mid \langle q_0 Z_0 q_1\rangle$$

Nyní přidávejme pravidla pro $\langle q_0 Z_0 q_0\rangle$ a $\langle q_0 Z_0 q_1\rangle$. *Díky* $q_0 Z_0 \xrightarrow{a} q_0 X Z_0 \in \delta$ *přidáme*

$$\langle q_0 Z_0 q_0\rangle \to a\langle q_0 X q_0\rangle\langle q_0 Z_0 q_0\rangle \mid a\langle q_0 X q_1\rangle\langle q_1 Z_0 q_0\rangle\ ,$$
$$\langle q_0 Z_0 q_1\rangle \to a\langle q_0 X q_0\rangle\langle q_0 Z_0 q_1\rangle \mid a\langle q_0 X q_1\rangle\langle q_1 Z_0 q_1\rangle\ .$$

Opakováním tohoto postupu pro nově vznikající neterminály celkem ještě přidáme

$\langle q_0 X q_0\rangle \to a\langle q_0 X q_0\rangle\langle q_0 X q_0\rangle \mid a\langle q_0 X q_1\rangle\langle q_1 X q_0\rangle$, *a též*
$\langle q_0 X q_1\rangle \to a\langle q_0 X q_0\rangle\langle q_0 X q_1\rangle \mid a\langle q_0 X q_1\rangle\langle q_1 X q_1\rangle$, *protože* $q_0 X \xrightarrow{a} q_0 X X \in \delta$,
$\langle q_0 X q_1\rangle \to b$, *protože* $q_0 X \xrightarrow{b} q_1 \in \delta$
$\langle q_1 Z_0 q_1\rangle \to \varepsilon$, *protože* $q_1 Z_0 \xrightarrow{\varepsilon} q_1 \in \delta$,
$\langle q_1 X q_1\rangle \to \varepsilon \mid b$, *protože* $q_1 X \xrightarrow{\varepsilon} q_1$, $q_1 X \xrightarrow{b} q_1 \in \delta$.

Poznamenejme, že neexistují žádná pravidla pro $\langle q_1 X q_0\rangle$ a $\langle q_1 Z_0 q_0\rangle$; *tyto se ovšem na pravých stranách v* P *vyskytují, tedy tato pravidla nemohou vygenerovat žádný terminální řetěz, tj. i příslušné levostranné neterminály* $\langle q_0 X q_0\rangle$ a $\langle q_0 Z_0 q_0\rangle$ *jsou nenormované (nepoužitelné). Pak ekvivalentní reduková gramatika má těchto sedm pravidel:*

1.	$S \to \langle q_0 Z_0 q_1\rangle$		
2.	$\langle q_0 Z_0 q_1\rangle \to a\langle q_0 X q_1\rangle\langle q_1 Z_0 q_1\rangle$	5.	$\langle q_1 Z_0 q_1\rangle \to \varepsilon$
3.,4.	$\langle q_0 X q_1\rangle \to a\langle q_0 X q_1\rangle\langle q_1 X q_1\rangle \mid b$	6.,7.	$\langle q_1 X q_1\rangle \to \varepsilon \mid b$

Příklad 3.54. *Nechť je dán PDA* $\mathcal{P} = (\{p, q, r, s\}, \{a, b, c, d\}, \{X\}, \delta, p, X, \emptyset)$, *kde* δ *je dána takto:* $pX \xrightarrow{a} pXX$, $pX \xrightarrow{b} r\varepsilon$, $pX \xrightarrow{c} q\varepsilon$, $qX \xrightarrow{d} sX$, $sX \xrightarrow{d} q\varepsilon$, $rX \xrightarrow{d} r\varepsilon$. *Pro* $\mathcal{P}$ *zkonstruujte část přechodového grafu (např. z* pX *alespoň dva* a*-přechody vedoucí postupně do* pXX *a* $pXXX$ *a všechny* b, c, d*-přechody (a totální stavy) vedoucí postupně do akceptujících* $q\varepsilon, r\varepsilon$*). Nalezněte ekvivaletní CFG a zkonstruujte její přechodový graf.*

Na závěr této části uveďme ještě jedno tvrzení, které je sice již obsaženo v důsledku 3.52, avšak v jeho důkazu bude uvedena přímočará a velmi důležitá konstrukce, která (mimo jiné) ozřejmí, proč jsme definovali rozšířený PDA.

Lemma 3.55. (O nedeterministické syntaktické analýze zdola nahoru) *Nechť* $\mathcal{G}$ *je libovolná CFG, pak lze zkonstruovat rozšířený PDA* $\mathcal{R}$ *takový, že* $L(\mathcal{G}) = L(\mathcal{R})$.

Idea důkazu: Z důvodů čitelnosti (viz dále) budeme nyní vrchol zásobníku psát (v definici δ i v konfiguracích) vždy vpravo. $\mathcal{R}$ bude opět de facto jednostavový (druhý stav bude sloužit jen jako koncový).

Na rozdíl od důkazu analogického tvrzení pro obyčejný PDA (viz věta 3.47), budeme konstruovat rozšířený PDA $\mathcal{R}$ tak, aby simuloval pravé odvození vstupní věty, přesněji řečeno reverzi tohoto odvození: práci začne nad vstupní větou (a – v pricipu – s prázdným zásobníkem) a skončí s prázným vstupem a v zásobníku bude kořen gramatiky (odtud název *analýza zdola nahoru*. Pravou větnou formu αAy bude mít k dispozici tak, že αA (jako výsledek dosavadní práce nad x – již přečtenou částí vstupu) je obsah zásobníku (s A na vrcholu) a y je dosud nepřečtená (nezpracovaná) část vstupu. Tedy invarintem výpočtu bude vlastnost:

obsah zásobníku zřetězen se zbytkem vstupu v $\mathcal{R} =$ pravá větná forma v $\mathcal{G}$ (*)

či přesněji řečeno, začne-li $\mathcal{R}$ pracovat nad vstupem, který je větou z $L(\mathcal{G})$, pak:

αAy je obsah zásobníku zřetězen se zbytkem vstupu

<=> (**)

αAy je pravá větná forma.

Rozšířený PDA $\mathcal{R}$ má kroky dvojího typu: (1) může kdykoli číst do zásobníku vstupní symbol, (2) je-li na vrcholu zásobníku řetěz tvořící pravou stranu nějakého pravidla v $\mathcal{G}$, může nahradit v zásobníku tuto pravou stranu odpovídajícím levostranným neterminálem; ze vstupu nic nečte. Krok typu (2) nazveme *redukcí* podle příslušného pravidla. Automat $\mathcal{R}$ bude tedy (opět) nedeterministický, tj. bude hádat, kdy (postupně) číst symboly ze vstupu a kdy a jak redukovat podřetězy v pravých větných formách (vrcholové řetězy v zásobníku), a to počínaje vstupním slovem tak, aby bylo dosaženo kořene gramatiky, právě když vstup je větou v $\mathcal{G}$.

Důkaz. Nechť $\mathcal{G} = (N, \Sigma, P, S)$ a položme $\mathcal{R} = (\{q, r\}, \Sigma, N \cup \Sigma \cup \{\perp\}, \delta, q, \perp, \{r\})$, kde $\perp$ je nově přidaný symbol a kde δ je definována takto:

1. $\delta(q, a, \varepsilon) = \{(q, a)\}$ pro všechna $a \in \Sigma$,
2. je-li $A \to \alpha$, pak $\delta(q, \varepsilon, \alpha)$ obsahuje (q, A) ,
3. $\delta(q, \varepsilon, \perp S) = \{(r, \varepsilon)\}$.

Úmluva: každé níže uvedené odvození je pravým odvozením.

Nyní zformulujme a dokažme implikaci “$\Leftarrow$” v (**). Indukcí vzhledem k n tedy ukažme tvrzení (všimněme si, že $\Rightarrow^*$ jen říká, že αAy je pravá větná forma – viz úmluva):

$$S \Rightarrow^* \alpha Ay \Rightarrow^n xy \implies (q, xy, \perp) \vdash^* (q, y, \perp\alpha A) \qquad (1)$$

Báze, tj. $n = 0$ je triviální – $\mathcal{R}$ neudělá žádný krok.

Předpokládejme, že (1) platí pro všechna $n' < n$. Pak můžeme psát $\alpha Ay \Rightarrow \alpha\beta y \Rightarrow^{n-1} xy$. Mohou nastat 2 případy:

Je-li $\alpha\beta \in \Sigma^*$, pak $\alpha\beta = x$ a $(q, xy, \perp) \vdash^* (q, y, \perp\alpha\beta) \vdash (q, y, \perp\alpha A)$.

Je-li $\alpha\beta \notin \Sigma^*$, pak lze psát $\alpha\beta = \gamma Bz$, kde B je nejpravější neterminál. Podle indukčního předpokladu máme, že $S \Rightarrow^* \gamma Bzy \Rightarrow^{n-1} xy$ implikuje $(q, xy, \perp) \vdash^* (q, zy, \perp\gamma B)$.

Jelikož $(q, zy, \perp\gamma B) \vdash^* (q, y, \perp\gamma Bz)$ je možná posloupnost kroků (čtení do zásobníku), dostáváme celkem, že (1) platí. Konečně $(q, \varepsilon, \perp S) \vdash (r, \varepsilon, \varepsilon)$, a tedy $L(\mathcal{G}) \subseteq L(\mathcal{R})$.

Abychom ukázali obrácenou inklusi (tj. implikaci “$\Rightarrow$” v (**)), dokažme indukcí vzhledem k n tvrzení

$$(q, xy, \bot) \vdash\!\!\frac{n}{\quad} (q, y, \bot\alpha A) \implies \alpha A y \Rightarrow^* xy \tag{2}$$

Báze indukce, $n = 0$, platí triviálně.

Pro indukční krok předpokládejme, že (2) platí pro všechna $n < m$. Pokud symbol na vrcholu zásobníku je neterminál, pak jistě musel být poslední krok proveden na základě pravidla redukce (viz bod 2 v definici δ automatu $\mathcal{R}$) – na vstupu neterminály nejsou. Můžeme tedy psát

$$(q, xy, \bot) \vdash\!\!\frac{m-1}{\quad} (q, y, \bot\alpha\beta) \vdash\!\!\frac{}{\quad} (q, y, \bot\alpha A),$$

kde $A \to \beta$ je pravidlo z P. Pokud se v řetězu $\alpha\beta$ vyskytuje nějaký neterminál, pak dle indukčního předpokladu máme $\alpha\beta y \Rightarrow^* xy$. Celkem tedy dostáváme $\alpha A y \Rightarrow \alpha\beta y \Rightarrow^* xy$, čímž je (2) dokázána.

Nyní specializací (2) dostaneme, že $(q, w, \bot) \vdash\!\!\frac{*}{\quad} (q, \varepsilon, \bot S)$ implikuje $S \Rightarrow^* w$. Jelikož $\mathcal{R}$ akceptuje w jen pokud $(q, w, \bot) \vdash\!\!\frac{*}{\quad} (q, \varepsilon, \bot S) \vdash\!\!\frac{}{\quad} (r, \varepsilon, \varepsilon)$, dostáváme, že $L(\mathcal{R}) \subseteq L(\mathcal{G})$, a tedy celkem $L(\mathcal{R}) = L(\mathcal{G})$, čímž je důkaz ukončen. □

Příklad 3.56. *Mějme $\mathcal{G}_0$ s pravidly $E \to E{+}T \mid T,\ T \to T{*}F \mid F,\ F \to (E) \mid i$. Pak rozšířený PDA z lemmatu 3.55 je $\mathcal{P} = (\{q, r\}, \{+, *, (,), i\}, \{E, T, F, +, *, (,), i, \bot\}, \delta, q, \bot, \{r\})$, kde δ je definována takto (připomeňme, že vrchol zásobníku píšeme vpravo):*

q	$\xrightarrow{a}$	qa	$\forall a \in \{+, *, (,), i\}$	*dle bodu 1*
$qE{+}T$	$\xrightarrow{\varepsilon}$	qE		*dle bodu 2*
qT	$\xrightarrow{\varepsilon}$	qE		*dle bodu 2*
$qT{*}F$	$\xrightarrow{\varepsilon}$	qT		*dle bodu 2*
qF	$\xrightarrow{\varepsilon}$	qT		*dle bodu 2*
$q(E)$	$\xrightarrow{\varepsilon}$	qF		*dle bodu 2*
qi	$\xrightarrow{\varepsilon}$	qF		*dle bodu 2*
$q\bot E$	$\xrightarrow{\varepsilon}$	$r\varepsilon$		*dle bodu 3*

Všimněme si, že pokud bychom nepřijali konvenci, že vrchol zásobníku píšeme vpravo, museli bychom ve výše uvedené definici funkce δ místo pravé strany α pravidla z P psát méně přehledný zrcadlový obraz α, tj. α^{R}. Z těchže důvodů jsou v akceptujícím výpočtu pro vstupní slovo $i + i * i$ (viz obrázek 3.5) konfigurace psány ve tvaru trojic (stav, obsah zásobníku, vstup) s vrcholem zásobníku vpravo (a to i pro názornost ověření podmínky (*) z úvodu k důkazu lemmatu 3.55).

Poznámka 3.57. *Na obrázku 3.5 si všimněme, že posloupnost odpovídajících pravidel z $\mathcal{G}_0$ použitých při výpočtu v $\mathcal{P}$ je reverzí posloupnosti pravidel použitých při pravé derivaci vstupní věty v $\mathcal{G}_0$. Pokud bychom na základě této posloupnosti postupně konstruovali odpovídající derivační strom, povšimneme si, že konstrukce začíná u jeho (levých) listů a jde směrem ke kořenu; odtud název této techniky – nedeterministická syntaktická analýza zdola nahoru. Nedeterminismus se objevuje v těchto dvou základních variantách:*

(1) v konfiguracích, kdy PDA má na vrcholu pravou stranu nějakého pravidla a volí zda redukovat tuto pravou stranu na odpovídající neterminál nebo zda číst ze vstupu – viz výše např. $(q,\ \bot E + T,\ *i)$, kde lze nejen provést uvedené čtení symbolu '', ale též redukci dle $E \to E{+}T$). Tuto situaci nazýváme konfliktem typu* čtení versus redukce;

(2) v konfiguracích, kdy PDA má na vrcholu takový řetěz, že jsou možné alespoň

krok výpočtu			odpovídající pravidlo z $\mathcal{G}_0$ pro následující krok
$(q,\ \bot,\ i+i*i)$	$\vdash^{i} (q,\ \bot i,$	$+i*i)$	$F \to i$
	$\vdash^{\varepsilon} (q,\ \bot F,$	$+i*i)$	$T \to F$
	$\vdash^{\varepsilon} (q,\ \bot T,$	$+i*i)$	$E \to T$
	$\vdash^{\varepsilon} (q,\ \bot E,$	$+i*i)$	
	$\vdash^{+} (q,\ \bot E+,$	$i*i)$	
	$\vdash^{i} (q,\ \bot E+i,$	$*i)$	$F \to i$
	$\vdash^{\varepsilon} (q,\ \bot E+F,$	$*i)$	$T \to F$
	$\vdash^{\varepsilon} (q,\ \bot E+T,$	$*i)$	
	$\vdash^{*} (q,\ \bot E+T*,$	$i)$	
	$\vdash^{i} (q,\ \bot E+T*i,$	$\varepsilon)$	$F \to i$
	$\vdash^{\varepsilon} (q,\ \bot E+T*F,$	$\varepsilon)$	$T \to T*F$
	$\vdash^{\varepsilon} (q,\ \bot E+T,$	$\varepsilon)$	$E \to E+T$
	$\vdash^{\varepsilon} (q,\ \bot E,$	$\varepsilon)$	
	$\vdash^{\varepsilon} (r,\ \varepsilon$	$\varepsilon)$	

Obrázek 3.5: Výpočet automatu $\mathcal{P}$ z příkladu 3.56 pro vstupní slovo $i+i*i$

dvě různé redukce (tj. redukce podle různých pravidel); jako příklad může opět sloužit výše uvedená $(q,\ \bot E+T,\ *i)$, kde lze redukovat jak $E+T$ na E, tak i příponu tohoto řetězu, tj. T, na (shodou okolností opět) E. Tuto situaci nazývýme konfliktem typu redukce versus redukce.

Variantu (1) lze obecně charakterizovat existencí jak pravidla $A \to \alpha$, tak i $B \to \alpha\beta$ (spolu s existencí například $S \Rightarrow^* \gamma Az$ a $S \Rightarrow^* \gamma Bz$), kdežto pro (2) je typická existence pravidel $A \to \alpha$ a $B \to \beta\alpha$ (spolu například s $S \Rightarrow^* \gamma\beta Az$ a $S \Rightarrow^* \gamma Bz$). Současný výskyt obou typů konfliktů je samozřejmě v konfiguraci PDA též možný.

Co se nedeterminismu týče, je tedy vidět, že zatímco u analýzy shora dolů se omezuje na případ, kdy máme volit, kterou z pravých stran nahradit neterminál na vrcholu zásobníku, je u analýzy zdola nahoru jeho povaha poněkud košatější.

3.3 Vlastnosti bezkontextových jazyků

3.3.1 Uzávěrové vlastnosti

V této části uvedeme některé uzávěrové vlastnosti CFL a ukážeme, jak je lze použít k důkazu, že nějaký jazyk je, nebo není CFL. Díky vztahu CFG a PDA (viz část 3.2) lze v níže uvedených důkazech použít jak bezkontextových gramatik, tak i zásobníkových automatů. V dalším textu $\mathcal{L}_2$ značí třídu všech bezkontextových jazyků a připomeňme, že pokud není řečeno jinak, vždy předpokládáme, že gramatika je redukovaná.

Věta 3.58. *$\mathcal{L}_2$ je uzavřena vzhledem k operaci (1) sjednocení, (2) zřetězení, (3) iteraci a (4) pozitivní iteraci.*

Důkaz. Nechť CFL $L_i, i = 1, 2$ je generován CFG $\mathcal{G}_i = (N_i, \Sigma_i, P_i, S_i)$, tj. $L_i = L(\mathcal{G}_i)$. Bez újmy na obecnosti můžeme předpokládat, že $N_1 \cap N_2 = \emptyset$ (v opačném případě lze například k N_2 vytvořit abecedu dvojníků $N_2' = \{A' \mid A \in N_2\}$).

(1) Jazyk $L = L_1 \cup L_2$ je generován gramatikou $\mathcal{G} = (N_1 \cup N_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}, S)$, kde S je nový symbol. Pak každá derivace v $\mathcal{G}$ musí začínat použitím buď $S \to S_1$ nebo $S \to S_2$, přičemž podmínka $N_1 \cap N_2 = \emptyset$ zaručuje, že při použití $S \to S_1$ (resp. $S \to S_2$) lze v dalším derivování používat jen pravidla z P_1 (resp. P_2). Formální důkaz, že $L(\mathcal{G}) = L(\mathcal{G}_1) \cup L(\mathcal{G}_2)$ (tj. obě inkluse) je ponechán čtenáři.

(2) Jazyk $L = L_1.L_2$ je generován gramatikou $\mathcal{G} = (N_1 \cup N_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S)$, kde S je nový symbol.

(3) Jazyk $L = L_1^*$ je generován gramatikou $\mathcal{G} = (N_1 \cup \{S\}, \Sigma_1, P_1 \cup \{S \to SS_1 \mid \varepsilon\}, S)$, kde S je nový symbol.

(4) Jazyk $L = L_1^+$ je generován gramatikou $\mathcal{G} = (N_1 \cup \{S\}, \Sigma_1, P_1 \cup \{S \to SS_1 \mid S_1\}, S)$, kde S je nový symbol.

Formální důkazy ad (2) – (4) jsou opět ponechány čtenáři. □

Příklad 3.59. *Ukažme, že jazyk $L_1 = \{a^{m_1} b^{m_1} \ldots a^{m_n} b^{m_n} \mid n \geq 0, m_i \geq 0, 1 \leq i \leq n\}$ je CFL. Stačí si uvědomit, že L_1 lze obdržet iterací jazyka $L = \{a^m b^m \mid m \geq 0\}$, který je CFL, tj. $L_1 = L^*$.*

Věta 3.60. *$\mathcal{L}_2$ není uzavřena vzhledem k operacím (1) průniku a (2) doplňku.*

Důkaz. (1) Mějme jazyky $L_1 = \{a^n b^n c^m \mid m, n \geq 1\}$ a $L_2 = \{a^m b^n c^m \mid m, n \geq 1\}$. Oba tyto jazyky jsou CFL (například L_1 je generován CFG s pravidly $S \to S_1 C, S_1 \to aS_1 b \mid ab, C \to Cc \mid c$). Kdyby $\mathcal{L}_2$ byla uzavřena vzhledem k operaci průniku, pak i $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 1\}$ musel být bezkontextový, což však není – viz příklad 3.26.

(2) Neuzavřenost $\mathcal{L}_2$ vůči doplňku plyne okamžitě z její uzavřenosti na sjednocení, neuzavřenosti na průnik a z De Morganových pravidel: pro libovolné L_1, L_2 platí

$$L_1 \cap L_2 = \neg(\neg L_1 \cup \neg L_2),$$

tj., kdyby $\mathcal{L}_2$ byla uzavřena na doplněk, musela by být uzavřena i na průnik – spor. □

Věta 3.61. *$\mathcal{L}_2$ je uzavřena vzhledem k průniku s regulárním jazykem.*

Důkaz. $L \in \mathcal{L}_2$ značí, že existuje PDA $\mathcal{P} = (Q_1, \Sigma, \Gamma, \delta_1, q_0^1, Z_0, F_1)$, takový, že $L = L(\mathcal{P})$ a R regulární značí, že existuje (bez újmy na obecnosti deterministický) konečný automat $\mathcal{A} = (Q_2, \Sigma, \delta_2, q_0^2, F_2)$, takový, že $R = L(\mathcal{A})$. Abychom ukázali, že $L' = L \cap R \in \mathcal{L}_2$, sestrojme PDA $\mathcal{P}'$, takový, že $L' = L(\mathcal{P}')$.

Položme $\mathcal{P}' = (Q', \Sigma, \Gamma, \delta', q_0', Z_0, F')$, kde

1. $Q' = Q_1 \times Q_2$,
2. $q_0' = \langle q_0^1, q_0^2 \rangle$
3. $F' = F_1 \times F_2$ a
4. δ' : $\delta'(\langle p, q \rangle, a, Z)$ obsahuje $(\langle p', q' \rangle, \gamma) \stackrel{def}{\Longleftrightarrow}$ $\delta_1(p, a, Z)$ obsahuje (p', γ) a $\delta_2(q, a) = q'$

Zřejmě platí $w \in L(\mathcal{P}') \iff w \in L(\mathcal{P}) \cap L(\mathcal{R})$. □

Příklad 3.62. *Ukažme, že* $L = \{w \in \{a,b,c\} \mid w$ *obsahuje stejný počet symbolů* $a,b,c\}$ *není CFL. K tomu stačí uvážit, že* $L \cap a^*b^*c^* = \{a^nb^nc^n \mid n \geq 0\}$ *není CFL, a tedy ani* L *není CFL.*

Věta 3.63. *$\mathcal{L}_2$ je uzavřena vzhledem k substituci.*

Důkaz. Nechť $L \subseteq \Sigma^*$ je CFL a σ taková substituce, že pro každé $a \in \Sigma$ platí, že $\sigma(a) = L_a$ je CFL. Nechť $L = L(\mathcal{G})$ a $L_a = L(\mathcal{G}_a)$ pro každé $a \in \Sigma$. Bez újmy na obecnosti předpokládejme, že abecedy neterminálů všech uvedených CFG jsou vzájemně po dvou disjunktní a konstruujme gramatiku $\mathcal{G}'$ takto:

1. neterminály gramatiky $\mathcal{G}'$ jsou sjednocením neterminálů gramatiky $\mathcal{G}$ a neterminálů všech $\mathcal{G}_a$,
2. terminály v $\mathcal{G}'$ jsou sjednocením terminálů všech $\mathcal{G}_a$,
3. pravidla v $\mathcal{G}'$ jsou sjednocením pravidel všech $\mathcal{G}_a$ spolu s pravidly, která vzniknou takto: ke každému $A \to \alpha$ v $\mathcal{G}$ vytvoříme nové pravidlo, které vznikne tak, že v α každý výskyt terminálu a nahradíme neterminálem S_a – kořenem gramatiky $\mathcal{G}_a$ a
4. kořenem $\mathcal{G}'$ je kořen $\mathcal{G}$.

Formální důkaz je opět ponechán čtenáři. □

Příklad 3.64. *Nechť L je množina slov, která obsahují stejný počet výskytů symbolu a a b. Nechť dále* $L_a = \{0^n1^n \mid n \geq 1\}$ *a* $L_b = \{ww^R \mid w \in (0+2)^*\}$.
L lze generovat CFG $\mathcal{G}$ s pravidly

$$S \to aSbS \mid bSaS \mid \varepsilon,$$

L_a lze generovat CFG $\mathcal{G}_a$ s pravidly

$$S_a \to 0S_a1 \mid 01 \text{ a}$$

L_b lze generovat CFG $\mathcal{G}_b$ s pravidly

$$S_b \to 0S_b0 \mid 2S_b2 \mid \varepsilon.$$

Je-li f taková substituce, že $f(a) = L_a$ a $f(b) = L_b$, pak $f(L)$ lze generovat CFG s pravidly

$$S \to S_aSS_bS \mid S_bSS_aS \mid \varepsilon \qquad S_a \to 0S_a1 \mid 01 \qquad S_b \to 0S_b0 \mid 2S_b2 \mid \varepsilon.$$

Jelikož $\{a,b\}, \{ab\}, a^*$ a a^+ jsou CFL, pak uzavřenost $\mathcal{L}_2$ na substituci implikuje uzavřenost na sjednocení, zřetězení a (pozitivní) iteraci: sjednocení $L_a \cup L_b$ je substitucí L_a a L_b do $\{a,b\}$, podobně zřetězení $L_a.L_b$ je substitucí L_a a L_b do $\{ab\}$ a L_a^* je substitucí L_a do a^*. Tvrzení věty 3.58 bylo tedy možné prezentovat jako důsledek věty 3.63.

Důsledek 3.65. *$\mathcal{L}_2$ je uzavřena vůči (1) homomorfismu a (2) inverznímu homomorfismu.*

Důkaz. (1) Homomorfismus je speciálním případem substituce, vůči níž je $\mathcal{L}_2$ uzavřena.
(2) Důkaz uzavřenosti $\mathcal{L}_2$ na inverzní homomorfismus plyne z uzavřenosti $\mathcal{L}_2$ na substituci, homomorfismus a průnik s regulárním jazykem takto: Mějme libovolný homomorfismus $h : \Sigma \to \Delta^*$ a k Σ definujme abecedu dvojníků $\overline{\Sigma}$. Dále definujme:

1. substituci σ : $\sigma(c) = \overline{\Sigma}^* c \Sigma^*$ pro každé $c \in \Delta$,
2. regulární jazyk $L_1 = \{\overline{a}w \mid \overline{a} \in \overline{\Sigma}, w \in \Delta^*, h(a) = w\}$ a

3. homomorfismus $h_1 : (\overline{\Sigma} \cup \Delta) \to \Sigma^*$ takto: $h_1(\overline{a}) = a$ pro všechna $\overline{a} \in \overline{\Sigma}$
$h_1(c) = \varepsilon$ pro všechna $c \in \Delta$.

Zřejmě platí, že $h_1(\sigma(L) \cap L_1^*) = h^{-1}(L)$. Současně však z uzavřenosti třídy $\mathcal{L}_2$ vzhledem k výše zmíněným operacím (a faktu, že též jazyk L_1^* je regulární) plyne, že pro každý jazyk $L \in \mathcal{L}_2$ platí $h_1(\sigma(L) \cap L_1^*) \in \mathcal{L}_2$. Celkem tedy dostáváme $h^{-1}(L) \in \mathcal{L}_2$, což jsme měli dokázat. □

Příklad 3.66. *Mějme jazyk $L = \{ww \mid w \in \{a,b\}^*\}$ a ukažme, že L není CFL. Předpokládejme, že L je CFL. Pak ovšem $L_1 = L \cap a^+b^+a^+b^+$ by byl též CFL, ale $L_1 = \{a^ib^ja^ib^j \mid i,j \geq 1\}$, což je jazyk velmi podobný jazyku $L_2 = \{a^ib^jc^id^j \mid i,j \geq 1\}$ z příkladu 3.27 o němž jsme pomocí pumping lemmatu ukázali, že není CFL – pomocí obdobných argumentů lze totéž dokázat i o L_1.*

Pokud bychom k důkazu, že L_1 není CFL nechtěli použít pumping lemma, lze L_1 redukovat přímo na $L_2 = \{a^ib^jc^id^j \mid i,j \geq 1\}$ takto: Nechť h je homomorfismus definovaný takto: $h(a) = h(c) = a$ a $h(b) = h(d) = b$. Pak $h^{-1}(L_1)$ se skládá ze všech slov tvaru $x_1x_2x_3x_4$ takových, že x_1 a x_3 jsou z $\{a,c\}^+$ a mají stejnou délku a podobně x_2 a x_4 jsou z $\{b,d\}^+$ a mají stejnou délku. Pak ovšem $h^{-1}(L_1) \cap a^*b^*c^*d^* = L_2$, a tedy kdyby L_1 byl CFL, musel by L_2 být rovněž CFL, což (jak víme z příkladu 3.27) není, tudíž ani L_1 nemůže být CFL.

Příklad 3.67. *Na základě příkladu 3.66 lze ukázat, že jazyk fragmentů pascalských programů $L = \{$**begin var** w : integer; w := 0; **end** $\mid w \in \{a,b\}^+\}$ není bezkontextový. Nechť h je homomorfismus takový, že $h(a) = a$, $h(b) = b$ a $h(X) = \varepsilon$ v ostatních případech. Pak $h(L) = \{ww \mid w \in \{a,b\}^+\}$, a tedy L není CFL.*

V kompilátorech je tato situace řešena tak, že před vlastní syntaktickou analýzou je text zpracován lexikálním analyzátorem, který identifikátory, jejichž délka není v definici jazyka shora omezena, zpracovává tak, že je redukuje na dvojici fixní délky, kde první komponenta (kterou bere v potaz syntaktický analyzátor) udává, že se jedná o syntaktickou kategorii „identifikátor“; druhá komponenta obsahuje odkaz do tabulky, kde je skutečný textový tvar identifikátoru uložen.

3.3.2 Rozhodnutelné vlastnosti a regularita

Již v předchozích částech jsme ukázali, že existuje algoritmus, který pro libovolnou danou CFG $\mathcal{G}$ rozhoduje, zda $L(\mathcal{G}) = \emptyset$ či nikoliv (tzv. problém prázdnosti jazyka je pro CFLs tedy algoritmicky řešitelný – rozhodnutelný).

Další důležitou vlastnost jsme ukázali tím, že (opět) k libovolné dané CFG $\mathcal{G}$ lze sestrojit (jazykově) ekvivaletní PDA; jinak řečeno, máme k dispozici (nedeterministický) algoritmus, který rozhoduje problém zda $w \in L(\mathcal{G})$ či nikoliv (tzv. problém příslušnosti slova je v třídě CFLs rozhodnutelný).

Některé další vlastnosti CFL jsme již též ukázali v předchozích částech, mimo jiné i tzv. Pumping lemma pro CFL, které nám v některých případech (spolu s eventuelním využitím uzávěrových vlastností) umožní dokázat, že daný jazyk není bezkontextový. I následující věty jsou de facto důsledkem Pumping lemmatu pro CFL.

Věta 3.68. *Ke každé CFG $\mathcal{G}$ lze sestrojit čísla m, n taková, že $L(\mathcal{G})$ je nekonečný právě když existuje slovo $z \in L(\mathcal{G})$ takové, že $m < |z| \leq n$.*

Důkaz. Předpokládejme, že $\mathcal{G}$ je v CNF. Pak dle Pumping lemmatu pro CFL (viz 3.24) existují čísla p, q s vlastnostmi popsanými v tomto lemmatu. Položme $m = p$ a $n = p + q$.

1. $\Longleftarrow$: Jestliže $z \in L(\mathcal{G})$ je takové slovo, že $p < |z| \leq p + q$, pak lze psát $z = uvwxy$ $(vx \neq \varepsilon)$ a $uv^iwx^iy \in L(\mathcal{G})$ pro všechna $i \geq 0$. Tedy $L(\mathcal{G})$ obsahuje nekonečně mnoho slov tvaru uv^iwx^iy, je tedy nekonečný.

2. $\Longrightarrow$: Nechť $L(\mathcal{G})$ je nekonečný. Pak obsahuje i nekonečně mnoho slov délky větší než p – tuto množinu slov označme M. Zvolme v M libovolné takové slovo z, které má minimální délku a ukažme, že musí platit $p < |z| \leq p + q$.

Sporem: kdyby $|z| > p + q$, pak (opět dle Pumping lemmatu pro CFL) lze z psát ve tvaru $z = uvwxy$, kde $vx \neq \varepsilon, |vwx| \leq q$ a $uv^iwx^iy \in L(\mathcal{G})$ pro všechna $i \geq 0$. Tedy specielně pro $i = 0$ máme, že $uwy \in L(\mathcal{G})$ a současně $|uwy| < |uvwxy|$. Přitom však (díky $|uvwxy| > p + q$ a $|vwx| \leq q$) platí, že $|uwy| > (p + q) - q = p$; tedy $uwy \in M$, což je spor s volbou z jako slova z M s minimální délkou. Celkem tedy musí být $|z| \leq p + q$. □

Poznámka 3.69. *Naše dosavadní poznatky o (ne)prázdnosti a (ne)konečnosti CFL lze sumarizovat takto:* existují algoritmy, které pro libovolný daný CFL L určí, zda L je
(a) prázdný, (b) konečný, nebo (c) nekonečný.

Algoritmus pro problém ad (a) byl podán v 3.9. Problém ad (b) resp. ad (c) lze rozhodovat tak, že postupně pro všechna slova w taková, že $p < |w| \leq p + q$ (a těch je jen konečně mnoho nad konečnou abecedou Σ) testujeme, zda $w \in L(\mathcal{G})$ či nikoliv (pomocí ekvivaletního PDA). Pokud nalezme w takové, že $w \in L(\mathcal{G})$, pak $L(\mathcal{G})$ je nekonečný, v opačném případě je konečný.

Následující vlastnost charakterizuje ty CFL, které nejsou regulární a po ní uvedené tvrzení ilustruje jednu z aplikací GNF.

Definice 3.70. Nechť $\mathcal{G} = (N, \Sigma, P, S)$ je CFG. Řekneme, že $\mathcal{G}$ má vlastnost sebevložení, jestliže existují $A \in N$ a $u, v \in \Sigma^+$ taková, že $A \Rightarrow^+ uAv$. CFL L má vlastnost sebevložení, jestliže každá gramatika, která jej generuje, má vlastnost sebevložení.

Věta 3.71. *CFL L má vlastnost sebevložení, právě když L není regulární.*

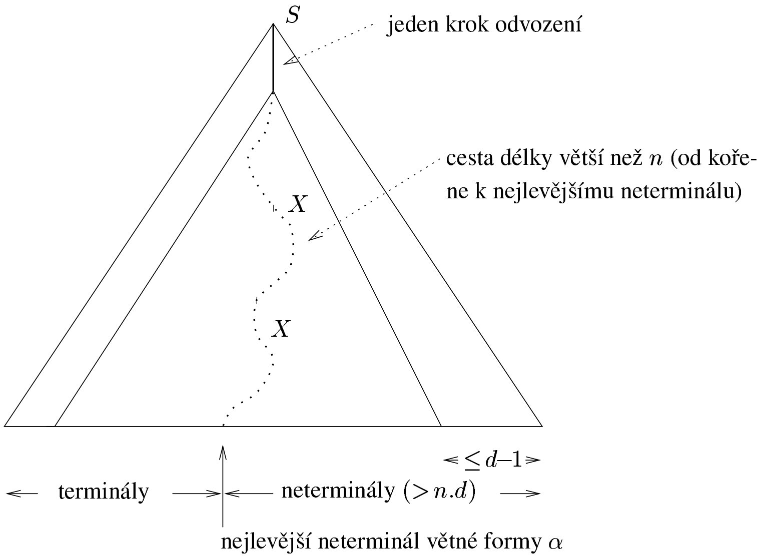
Důkaz. 1. $\Longrightarrow$ ($P \Longrightarrow Q$ ukazujeme jako $\neg Q \Longrightarrow \neg P$): Je-li L regulární, pak jej lze generovat regulární gramatikou a žádná regulární gramatika vlastnost sebevložení nemá.

2. $\Longleftarrow$ (opět $P \Longleftarrow Q$ ukazujeme jako $\neg Q \Longleftarrow \neg P$): Nechť L je generovatelný nějakou CFG, která nemá vlastnost sebevložení a ukažme, že pak L musí být regulární. Zmíněnou CFG lze převést na gramatiku $\mathcal{G} = (N, \Sigma, P, S)$ v GNF, která opět nemá vlastnost sebevložení a $L = L(\mathcal{G})$.

Označme $n = card(N)$ a $d = \max\{|\alpha|;\ A \to a\alpha \in P, A \in N\}$. Nyní ukažme platnost tohoto tvrzení:

v žádné levé větné formě v $\mathcal{G}$ nemůže být více než $n \cdot d$ výskytů neterminálů. (*)

Abychom to ukázali, předpokládejme opak, tj. existuje α takové, že $S \Rightarrow^* \alpha$ je levá



Obrázek 3.6: Derivační strom větné formy α – gramatika v GNF

derivace v $\mathcal{G}$ a α obsahuje více než $n \cdot d$ výskytů neterminálů. Označme C cestu z kořene S do nejlevějšího neterminálu v α. Z cesty C vyberme podposloupnost C' tvořenou jen těmi vrcholy, které odpovídají použití pravidel tvaru $A \to a\gamma$, kde $|\gamma| > 1$. Při každém odvozovacím kroku v podposloupnosti C' se zvýší počet výskytů neterminálů, a to nejvýše o $d-1$, což pro naše α značí, že v derivačním stromu pro α (viz obrázek 3.6) má C' délku větší než n. To však znamená, že alespoň dva z vrcholů na C' musí být označeny týmž neterminálem, řekněmě X. To ovšem značí, že $X \Rightarrow^+ uXv$, kde u i v jsou neprázdná, což je však spor s tím, že $\mathcal{G}$ nemá vlastnost sebevložení. Tím jsme dokázali tvrzení (*).

Jestliže se tedy v libovolné levé větné formě v $\mathcal{G}$ vyskytuje nejvýše $n \cdot d$ neterminálů, pak $L(\mathcal{G})$ lze generovat regulární gramatikou $\mathcal{G}' = (N', \Sigma, P', S')$, kde

- $N' = \{\langle\alpha\rangle \mid \alpha \in N^*, |\alpha| \leq n \cdot d\}$,
- $S' = \langle S \rangle$ a
- je-li $A \to a\alpha \in P, a \in \Sigma, \alpha \in N^*$, pak $\langle A\beta\rangle \to a\langle\alpha\beta\rangle \in P'$ pro každé β takové, že $|\alpha\beta| \leq n \cdot d$.

Je zřejmé, že $L(\mathcal{G}) = L(\mathcal{G}')$ a $\mathcal{G}'$ je regulární gramatika. $\square$

Zdůrazněme, že existence tvrzení o vlastnosti sebevložení charakterizující ty CFLs, které nejsou regulární, rozhodně neimplikuje existenci algoritmu, který by uměl pro libovolný daný CFL jazyk L rozhodovat, zda L tuto vlastnost má, či nemá; v dalším textu bude ukázáno, že neexistuje takový algoritmus, který by pro L rozhodoval, zda existuje regulární jazyk R takový, že $L = R$. Dokonce neexistuje ani algoritmus, který by rozhodoval, zda $L = \Sigma^*$.

3.4 Deterministické zásobníkové automaty

3.4.1 Definice DPDA a jejich základní vlastnosti

Definice 3.72. Řekneme, že PDA $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je *deterministický* (DPDA), jestliže jsou splněny tyto podmínky:

1. pro všechna $q \in Q$ a $Z \in \Gamma$ platí: kdykoliv $\delta(q, \varepsilon, Z) \neq \emptyset$, pak $\delta(q, a, Z) = \emptyset$ pro všechna $a \in \Sigma$;
2. pro žádné $q \in Q, Z \in \Gamma$ a $a \in \Sigma \cup \{\varepsilon\}$ neobsahuje $\delta(q, a, Z)$ více než jeden prvek.

Řekneme, že L je *deterministický bezkontextový jazyk* (DCFL, stručněji též deterministický jazyk), právě když existuje DPDA $\mathcal{M}$ takový, že $L = L(\mathcal{M})$.

Podmínka 1 vylučuje možnost volby mezi krokem nezávislým na vstupním symbolu (ε-krokem) a krokem, kdy se ze vstupu čte. Podmínka 2 říká, že jak v případě čtecího kroku, tak i pro ε-krok, neexistuje více než jedna varianta, jak dále pokračovat.

Lemma 3.73. *Ke každému DPDA $\mathcal{M}$ lze sestrojit DPDA $\mathcal{N}$ takový, že $L_e(\mathcal{M}) = L(\mathcal{N})$*

Důkaz. Tvrzení se dokáže stejně jako analogické tvrzení věty 3.39, část 2. ($\Longrightarrow$). Z uvedené konstrukce je okamžitě vidět, že je-li daný $\mathcal{M}$ deterministický, pak i $\mathcal{N}$ simulující jeho činnost je deterministický. □

Poznámka 3.74. *Poznamenejme, že obrácené tvrzení k tvrzení 3.73 obecně neplatí (důvody, jak uvidíme, jsou čistě technického, formálního charakteru). Po vyprázdnění zásobníku nemůže totiž žádný PDA (a tedy i DPDA) pokračovat ve výpočtu. Máme-li tedy dán takový jazyk L, že existuje slovo $u \in L$ a též $uv \in L, v \neq \varepsilon$, pak každý DPDA $\mathcal{M}$ akceptující prázdným zásobníkem musí po přečtení u akceptovat zastavením s prázdným zásobníkem. Tatáž situace však nastane, dáme-li automatu $\mathcal{M}$ na vstup slovo uv: pak $\mathcal{M}$ toto slovo nemůže dočíst do konce, a tedy $\mathcal{M}$ neakceptuje uv, tedy nerozpoznává L. Příkladem takového jazyka je dokonce konečný (tedy v Chomského hierarchii regulární) jazyk $\{a, aa\}$. Pokud naopak L výše uvedenou vlastnost nemá (tj. neexistuje slovo $u \in L$ takové, že rovněž $uv \in L, v \neq \varepsilon$), pak říkáme, že L je* bezprefixový*; z tohoto důvodu tedy DPDA akceptující prázdným zásobníkem nemohou rozpoznávat jazyky, které nejsou bezprefixové. Poznamenejme, že $\{a^n b^n \mid n \geq 1\}$ je příkladem bezprefixového CFL – tedy rozpoznatelný DPDA prázdným zásobníkem. Zkonstruujte jej! (srv. příklad 3.41)*

Je zřejmé, že ani technika "testu dna zásobníku" problém v případě DPDA neřeší – determinismus neumožňuje hádat, zda byl již přečen poslední symbol ze vstupu. Povšimněme si však, že pro každý $L \subseteq \Sigma^*$ a $\dashv \notin \Sigma$ je jazyk $L.\{\dashv\}$ bezprefixový (symbol $\dashv$ hraje roli ***eof****, pokud vstup chápeme jako soubor – file).*

V dalším textu tedy budeme u každého DPDA předpokládat, pokud nebude řečeno jinak, že ke konci vstupního slova je přidán znak (tzv. pravá koncová značka*)$\dashv \notin \Sigma$. Formální definici DPDA s pravou koncovou značkou (na vstupu) přenecháváme čtenáři jako cvičení.*

Definice 3.75. Řekneme, že rozšířený PDA $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je *deterministický* (DPDA) jestliže jsou splněny tyto podmínky:

1. Pro žádná $q \in Q, a \in \Sigma \cup \{\varepsilon\}$ a $\gamma \in \Gamma^*$ neplatí $card(\delta(q, a, \gamma)) > 1$.

2. Je-li $\delta(q, a, \alpha) \neq \emptyset, \delta(q, a, \beta) \neq \emptyset$ a $\alpha \neq \beta$, pak ani α není předponou β ani β není předponou α.
3. Je-li $\delta(q, a, \alpha) \neq \emptyset$ a $\delta(q, \varepsilon, \beta) \neq \emptyset$, pak ani α není předponou β ani β není předponou α.

Poznamenejme, že podmínky 2 a 3 jsou formulovány vzhledem ke konvenci, že v konfiguracích píšeme vrchol zásobníku vlevo, tj. α, β jsou vrcholové řetězy v zásobníku. Při konvenci s vrcholem zásobníku vpravo bychom museli v podmínkách 2 a 3 místo "předpona" použít "přípona".

Z definice 3.75 je vidět, že ve speciálním případě, kdy rozšířený PDA je "obyčejným" PDA, uvedená definice souhlasí s definicí 3.72. Poznamenejme též, že pokud je konstrukce z důkazu lemmatu 3.45 aplikována na rozšířený PDA $\mathcal{P}$, pak výsledkem bude DPDA když a jen když $\mathcal{P}$ je rozšířeným DPDA.

Definice 3.76. (normální forma (D)PDA) Řekneme, že DPDA $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je v normální formě jestliže platí: je-li $\delta(q, a, X) = (p, \gamma)$, pak buď (a) $\gamma = \varepsilon$ nebo (b) $\gamma = X$ nebo (c) $\gamma = YX$ pro nějaké $Y \in \Gamma$.

Identicky lze definovat normální formu i pro (nedeterministický) PDA. Uvedená normální forma tedy požaduje, aby jediné povolené operace nad zásobníkem byly odstranění vrcholového symbolu ze zásobníku, někdy též zvaná *pop* (viz podmínka (a)) nebo přidání jednoho symbolu, řekněme Z, na vrchol zásobníku, někdy též značená *push (Z)* (viz podmínka (c)); podmínka (b) povoluje změnit pouze vnitřní stav, a to bez změny zásobníku.

Následující dvě lemmata dokazují, že k libovolnému DPDA (resp. i PDA) lze sestrojit ekvivaletní DPDA (resp. PDA) v normální formě (důkazy pro PDA jsou kopiemi důkazů pro DPDA a nejsou tedy uvedeny). První lemma říká, že v žádném kroku DPDA nemusí ukládat na zásobník více než jeden symbol, protože namísto uložení řetězu symbolů je možné tento řetěz uložit na zásobník postupně, symbol po symbolu, a to pomocí ε-kroků. Navazující druhé lemma již konstruuje DPDA v normální formě, tj. ukazuje navíc, že DPDA nikdy nemění vrcholový symbol – nanejvýš nad něj jeden symbol přidá (tedy pokud vrcholový symbol není přímo odstraněn). Těmto změnám vrcholového symbolu se vyhneme tím, že DPDA si bude pamatovat vrcholový symbol v konečně stavové řídicí jednotce a požadované změny se budou odehrávat pouze v ní.

Lemma 3.77. *Ke každému DPDA $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ existuje s ním ekvivaletní DPDA $\mathcal{N} = (Q', \Sigma, \Gamma, \delta', q_0, Z_0, F)$ takový, že je-li $\delta'(q, a, X) = (p, \gamma)$, pak $|\gamma| \leq 2$.*

Důkaz. Je-li $\delta(q, a, X) = (r, \gamma)$ takové, že $|\gamma| > 2$, označme $\gamma = Y_1 \dots Y_n$, pro nějaké $n \geq 3$. Přidejme nové nekoncové stavy $p_1, \dots, p_{n-2}$ a δ rozšiřme na δ' takto:

$$\begin{array}{rcll} \delta'(q, a, X) & = & (p_1, Y_{n-1}Y_n), & \\ \delta'(p_i, \varepsilon, Y_{n-i}) & = & (p_{i+1}, Y_{n-i-1}, Y_{n-i}) & \text{pro } 1 \leq i \leq n-3 \text{ a} \\ \delta'(p_{n-2}, \varepsilon, Y_2) & = & (r, Y_1Y_2) \; . & \end{array}$$

Automaty $\mathcal{M}$ a $\mathcal{N}$ jsou evidentně ekvivaletní – jediný rozdíl je v tom, že pokud $\mathcal{N}$ při čtení symbolu a ve stavu q má X na vrcholu zásobníku nahradit řetězem $\gamma = Y_1 \dots Y_n$ a nabýt stavu r, pak to neučiní v jednom kroku, ale v $n - 1$ krocích. □

Lemma 3.78. *Ke každému DCFL L existuje DPDA $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ v normální formě takový, že $L = L(\mathcal{M})$.*

Důkaz. Nechť $L = L(\mathcal{M}')$ pro nějaký DPDA $\mathcal{M}' = (Q', \Sigma, \Gamma', \delta', q_0', X_0, F')$ splňující podmínky kladené na automat $\mathcal{N}$ v lemmatu 3.77. Sestrojíme $\mathcal{M}$, který bude simulovat činnost $\mathcal{M}'$ tak, že vrcholový symbol na zásobníku simulovaného automatu $\mathcal{M}'$ se bude uchovávat v množině stavů simulujícího automatu $\mathcal{M}$.

Položme $Q = Q' \times \Gamma'$, $q_0 = [q_0', X_0]$, $F = F' \times \Gamma'$, $\Gamma = \Gamma' \cup \{Z_0\}$, kde Z_0 je nový symbol a definujme δ takto:

1. je-li $\delta'(q, a, X) = (p, \varepsilon)$, pak $\forall Y \in \Gamma'$ položme $\delta([q, X], a, Y) = ([p, Y], \varepsilon)$,
2. je-li $\delta'(q, a, X) = (p, Y)$, pak $\forall Z \in \Gamma'$ položme $\delta([q, X], a, Z) = ([p, Y], Z)$,
3. je-li $\delta'(q, a, X) = (p, YZ)$, pak $\forall W \in \Gamma'$ položme $\delta([q, X], a, W) = ([p, Y], ZW)$.

Bod 1 říká, že pokud $\mathcal{M}'$ odstraní vrcholový symbol, pak $\mathcal{M}$ též odstraní vrcholový symbol s tím, že nový vrchol si zapamatuje ve své množině stavů. Bod 2 říká, že pokud $\mathcal{M}'$ změní svůj vrcholový symbol, pak $\mathcal{M}$ zaznamenává tuto změnu pouze v množině stavů. Konečně 3 říká, že pokud zásobník u $\mathcal{M}'$ roste, pak $\mathcal{M}$ si odloží příslušný symbol na svůj zásobník.

Nyní se snadno ověří (indukcí vzhledem k počtu kroků automatu), že platí:

$(q_0', w, X_0) \vdash^*_{\mathcal{M}'} (q, \varepsilon, X_1 X_2 \ldots X_n) \iff$

$([q_0', X_0], w, Z_0) \vdash^*_{\mathcal{M}} ([q, X_1], \varepsilon, X_2 X_3 \ldots X_n Z_0)$. Tedy $L(\mathcal{M}) = L(\mathcal{M}')$. □

3.4.2 Uzávěrové vlastnosti deterministických jazyků

V řadě (i praktických) aplikací je třeba zjistit, zda daný jazyk L je (anebo není) DCFL. K důkazu, že je DCFL stačí nalézt odpovídající DPDA (alternativně lze zkonstruovat nějakou tzv. LR gramatiku, kterými se zde však nezabýváme). Obrácená situace, kdy chceme ukázat, že L není DCFL, může být složitější. Pokud by L nebyl ani CFL, můžeme použít pumping lemma, ale často L může být CFL, ale ne DCFL. Jelikož není známo žádné pumping lemma, které by platilo specielně pro DCFL, musíme se spolehnout jen na uzávěrové vlastnosti. Naštěstí DCFL jsou uzavřeny na některé operace, například vůči doplňku, na něž CFL obecně uzavřeny nejsou. V dalším textu se na třídu všech deterministických bezkontextových jazyků odvoláváme jako na "třídu DCFL".

Věta 3.79. *Třída DCFL je uzavřena vzhledem k průniku s regulárním jazykem.*

Důkaz. Konstrukce DPDA akceptujícího průnik CFL L a regulárního jazyka R je identická s konstrukcí z důkazu věty o uzavřenosti třídy CFL vůči průniku s regulárním jazykem (viz 3.61) – je-li $\mathcal{M}$ rozpoznávající L deterministický, je též determistickým i konstruovaný zásobníkový automat rozpoznávající $L \cap R$. □

Věta 3.80. *Třída DCFL není uzavřena vzhledem k operaci průniku.*

Důkaz. Snadno se nahlédne, že jazyky L_1 a L_2 v části (1) důkazu věty 3.60 o neuzavřenosti CFL vůči průniku jsou deterministické. □

Nyní budeme chtít ukázat, že třída všech DCFL je uzavřena vůči operaci doplňku. V důkazu téže vlastnosti pro regulární jazyky jsme jednoduše záměnili koncové a nekoncové stavy v odpovídajícím deterministickém konečném automatu. Avšak u DPDA tato technika není takto přímočaře použitelná, a to hned ze dvou důvodů.

1. DPDA $\mathcal{M}$ nemusí vždy dočíst vstupní slovo až do konce, a to z těchto důvodů:
   (a) $\mathcal{M}$ se dostane do takové konfigurace, že další krok není definován nebo
   (b) $\mathcal{M}$ se dostane do nekonečného cyklu ε-kroků, kdy pouze pracuje s (konečným) obsahem zásobníku, aniž by četl ze vstupní pásky. Jinak řečeno, dělá pouze ε-kroky, přičemž buď
      i. jeho zásobník neomezeně roste nebo
      ii. zásobník neroste neomezeně, ale jeho obsah "cyklí", tj. po jistém počtu kroků se jeho obsah opakuje.

   Pokud $\mathcal{M}$ nedočte slovo až do konce, pak toto slovo není akceptováno a potíž v těchto případech spočívá v tom, že po pouhé záměně koncových a nekoncových stavů by takto modifikovaný DPDA opět slovo nedočetl, tj. nepřijal.
2. DPDA $\mathcal{M}$ sice dočte slovo až do konce, ale pak může ještě provést jistou posloupnost ε-kroků, při které prochází jak nekoncovými, tak i koncovými stavy. Zde je problém v tom, že pokud $\mathcal{M}$ takto akceptuje, pak po záměně koncových a nekoncových stavů by modifikovaný DPDA opět slovo akceptoval.

Následující lemma 3.81 odstraňuje problémy typu 1, problém 2 je řešen ve větě 3.82.

Lemma 3.81. *Ke každému DPDA $\mathcal{M}$ existuje ekvivalentní DPDA $\mathcal{M}$', který přečte každé vstupní slovo až do konce.*

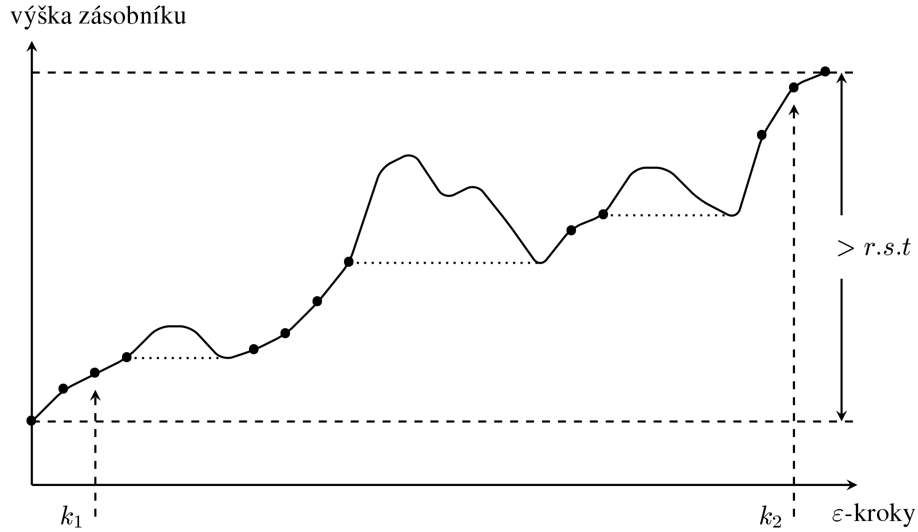
Důkaz. Potíž ad 1a odstraníme celkem snadno. Pokud by $\mathcal{M}$ nedočetl slovo proto, že vyprázdní svůj zásobník, použijeme techniku "test dna zásobníku" (viz důkaz věty 3.39 a poznámka 3.40), tj. $\mathcal{M}$' bude mít své vlastní dno Z_0' nad nějž si uloží dno Z_0 simulovaného automatu. Jestliže $\mathcal{M}$ vyprázdní svůj zásobník, $\mathcal{M}$' je schopen číst své lokální dno Z_0' a na základě toho přejde do nově přidaného (nekoncového) stavu d, v němž dočte vstup až do konce. Rovněž pro každou kombinaci stavu, vstupního symbolu (včetně ε) a vrcholu zásobníku, pro kterou není definován další krok, dodefinujeme tento krok přechodem do stavu d.

Problémy ad 1b vyžadují poněkud více úsilí. Označme $s = \text{card}(Q)$, $t = \text{card}(\Gamma)$ a $r = \max\{|\gamma|;\ \delta(p,\varepsilon,Z) = (p',\gamma), p, p' \in Q, Z \in \Gamma\} - 1$. Tedy r udává maximum, o kolik symbolů se může zásobník zvětšit při jednom ε-kroku. Nyní ukažme platnost tohoto tvrzení:

zásobník neomezeně roste při ε-krocích $\iff$ (*)
během nějaké posloupnosti ε-kroků se jeho délka zvětší o více než $r.s.t$

(*) $\Longrightarrow$: evidentní (roste-li zásobník nade všechny meze, pak vzroste i o více než $r.s.t$).
(*) $\Longleftarrow$: jestliže během nějaké posloupnosti ε-kroků zásobník povyroste o více než $r.s.t$, pak lze z této posloupnosti konfigurací vybrat (pod)posloupnost všech konfigurací takových, že při přechodu od i-té k $i+1$-ní konfiguraci zásobník vzroste nad úroveň, kterou měl v i-té konfiguraci (rostoucí posloupnost vzhledem k délce zásobníku). Tato vybraná posloupnost má jistě délku větší než $s.t$ (viz význam konstant r, s, t).

To ovšem značí, že mezi vybranými konfiguracemi existují alespoň dvě konfigurace k_1 a k_2, které se shodují svým stavem a svým vrcholovým symbolem, přičemž délka



Obrázek 3.7: Vybraná podposloupnost konfigurací při ε-krocích.

zásobníku je v k_2 větší než v k_1 – viz obrázek 3.7. Jelikož $\mathcal{M}$ je deterministický, pak posloupnost kroků, kterou prováděl mezi k_1 a k_2, bude dělat i nadále, a to ad infinitum. Tedy zásobník poroste nade všechny meze, čímž jsme dokázali tvrzení (*).

K detekci chování typu 1(b)i stačí tedy kontrolovat, zda se během nějaké nepřetržité posloupnosti ε-kroků nezvětší délka zásobníku o více než $r.s.t$. Tuto kontrolu zabezpečíme tak, že do konečně stavové řídicí jednotky přidáme čítač $c1$, který může nabývat (konečně mnoha) hodnot z intervalu $\langle 0, r.s.t \rangle$ a s nímž simulující $\mathcal{M}'$ pracuje takto:

- kdykoli se přečte symbol ze vstupu, $c1$ se nastaví na 0;
- při každém ε-kroku se $c1$ zvětší/zmenší o hodnotu, o kterou se zvětšila/zmenšila délka zásobníku;
- místo záporných čísel $c1$ nabývá hodnoty 0;
- pokud by měl $c1$ nabýt hodnoty větší než $r.s.t$, pak simulující $\mathcal{M}'$ přejde do "záchytného stavu d (nově přidaný nekoncový stav – viz tento důkaz, část ad 1a), v němž dočte vstup až do konce.

Tím je tedy odstraněn problém ad 1(b)i.

Nyní se zabývejme možnostmi detekce situací ad 1(b)ii, tj. když během nekonečné posloupnosti ε-kroků zásobník neroste neomezeně. Jinak řečeno, jeho délka nepřesáhne určitou mez. V průběhu této posloupnosti zásobník nemůže vzrůst o více než $r.s.t$, jinak by totiž rostl nade všechny meze – viz tvrzení (*). Nechť tedy při této posloupnosti ε-kroků zásobník nikdy neklesne pod úroveň h, a tedy nikdy nevzroste nad úroveň $h+r.s.t$. To však značí, že nejpozději za $s.(t+1)^{r.s.t}$ kroků se musí dostat do alespoň dvou konfigurací, které se shodují svým stavem i (celým) obsahem zásobníku nad hranicí h.

Pro detekci chování typu 1(b)ii tedy přidáme další čítač $c2$, který může nabývat hodnot z intervalu $\langle 0, s.(t+1)^{r.s.t} \rangle$ a který bude uchovávat počet provedených ε-kroků:

jestliže bude vynulován čítač $c1$, nastavíme na 0 i čítač $c2$ a při každém ε-kroku zvětšíme $c2$ o jedničku. Pokud by mělo dojít k přetečení $c2$, přejde simulující DPDA $\mathcal{M}'$ opět do záchytného stavu d, v němž dočte vstup až do konce.

Formální popis simulujícího $\mathcal{M}'$ rozšířeného oproti $\mathcal{M}$ o možnost testu dna (tj. nový počáteční stav, nové dno zásobníku Z_0' a záchytný stav d) a o čítače $c1$ a $c2$ je vynechán a lze jej nalézt v doporučené literatuře. □

Věta 3.82. *Třída deterministických bezkontextových jazyků je uzavřena vůči doplňku.*

Důkaz. Mějme libovolný DCFL L a bez újmy na obecnosti předpokládejme, že $L = L(\mathcal{M})$, pro nějaký DPDA $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ splňující podmínky lemmatu 3.81. Sestrojme DPDA $\mathcal{M}'$, který má rozpoznávat doplněk jazyka L.

Idea konstrukce: $\mathcal{M}'$ bude simulovat činnost $\mathcal{M}$, přičemž musíme mít na zřeteli problém 2 uvedený v diskusi před lemmatem 3.81. Stavy budou nyní dvojice z $Q \times \{p, n, f\}$, kde smyslem druhé komponenty ve stavu $[q, *]$ je zaznamenat mezi dvěma čtecími kroky, které mohou být proloženy posloupností ε-kroků, zda $\mathcal{M}$ prošel či neprošel koncovým stavem.

Pokud od posledního čtení vstupního symbolu $\mathcal{M}$ prošel při posloupnosti ε-kroků koncovým stavem, pak druhá komponenta $\mathcal{M}'$ je rovna p (prošel koncovým stavem), v opačném případě je n (neprošel koncovým stavem) – viz níže definice δ', bod 4a.

Simulaci, kdy $\mathcal{M}$ má číst vstupní symbol, popisuje bod 4b níže uvedené definice δ'. Má-li $\mathcal{M}'$ má ve své druhé komponentě příznak p, pak $\mathcal{M}'$ simuluje krok automatu $\mathcal{M}$ a v závislosti na tom, zda $\mathcal{M}$ se tímto krokem dostane do koncového či nekoncového stavu, $\mathcal{M}'$ aktualizuje svoji druhou komponentu na p či n. Pokud na vstupu již žádný symbol není, sdimulující $\mathcal{M}'$ končí ve stavu $[q, p] \notin F'$ pro nějaké q.

Je-li druhá komponenta rovna n, pak $\mathcal{M}'$ ji nejprve změní při ε-kroku z n na f. Poté $\mathcal{M}'$ opět simuluje krok automatu $\mathcal{M}$ a aktualizuje druhou komponentu na p či n v závislosti na tom, zda nový stav simulovaného $\mathcal{M}$ je či není koncový. Poznamenejme, že pokud na vstupu již žádný symbol není, $\mathcal{M}'$ skončí svoji činnosti ve stavu $[q, f] \in F'$ pro nějaké q.

Formálně definujme DPDA $\mathcal{M}' = (Q', \Sigma, \Gamma, \delta', q_0', Z_0, F')$, kde

1. $Q' = \{\, [q, *] \mid q \in Q, * \in \{p, n, f\}\,\}$,
2. $q_0' = \begin{cases} [q_0, p] & \text{je-li } q_0 \in F, \\ [q_0, n] & \text{je-li } q_0 \notin F, \end{cases}$
3. $F' = \{\, [q, f] \mid q \in Q\}$,
4. δ' je pro všechna $q, q' \in Q$ a $a \in \Sigma$ definována takto:
   (a) je-li $\delta(q, \varepsilon, Z) = (q', \gamma)$, pak
   $\delta'([q, p], \varepsilon, Z) = ([q', p], \gamma)$ a
   $$\delta'([q, n], \varepsilon, Z) = \begin{cases} ([q', p], \gamma) & \text{je-li } q' \in F, \\ ([q', n], \gamma) & \text{je-li } q' \notin F, \end{cases}$$
   (b) je-li $\delta(q, a, Z) = (q', \gamma),\ a \in \Sigma$, pak
   $\delta'([q, n], \varepsilon, Z) = ([q, f], Z)$ a
   $$\delta'([q, p], a, Z) = \delta'([q, f], a, Z) = \begin{cases} ([q', p], \gamma) & \text{je-li } q' \in F, \\ ([q', n], \gamma) & \text{je-li } q' \notin F. \end{cases}$$

Ukažme nyní, že $L(\mathcal{M}')$ je doplňkem $L(\mathcal{M})$. Nechť $a_1a_2 \ldots a_n \in L(\mathcal{M})$. Pak $\mathcal{M}$ vejde do koncového stavu (někdy) po přečtení a_n. V tomto případě bude druhá komponenta $\mathcal{M}'$ nastavena na p (a to předtím, než by $\mathcal{M}'$ mohl číst nějaký vstup za a_n). Tedy $\mathcal{M}'$ nebude akceptovat, tj. schopen vejít do stavu s druhou komponentou f (pokud a_n je posledním vstupním symbolem).

Je-li naopak $a_1a_2 \ldots a_n \notin L(\mathcal{M})$, pak (dle lemmatu 3.81) $\mathcal{M}'$ po přečtení a_n musí po jisté době přestat dělat ε-kroky a chtít číst další vstupní symbol. V této situaci je však druhá komponenta $\mathcal{M}'$ rovna n, protože $a_1a_2 \ldots a_n \notin L(\mathcal{M})$. Na základě pravidla 4b v definici δ' bude $\mathcal{M}'$ akceptovat (a to před eventuelním pokusem o čtení dalšího vstupního symbolu). Celkem tedy $L(\mathcal{M}') = \neg L(\mathcal{M})$. □

Důsledek 3.83. *Každý DCFL je akceptován nějakým DPDA, který v koncovém stavu nedělá žádné ε-kroky.*

Důkaz. Tvrzení je implicitně obsaženo v důkazu výše uvedené věty 3.82 – v žádném koncovém stavu (tj. s druhou komponentou rovnou f) není možný žádný ε-krok (δ' není pro tento případ definována). □

Důsledek 3.84. *Třída DCFL není uzavřena vzhledem ke sjednocení.*

Důkaz. Plyne okamžitě z uzavřenosti DCFL vůči doplňku (viz věta 3.82), neuzavřenosti vůči průniku (viz věta 3.80) a De Morganových pravidel. □

Příklad 3.85. *Uzavřenosti DCFL vůči doplňku lze někdy využít k důkazu, že daný CFL není deterministický.*

Ukažme, $L = \neg\{a^nb^nc^n \mid n \geq 1\}$ není DCFL. Rozborem po případech, kdy $w \notin \{a^nb^nc^n \mid n \geq 1\}$, se snadno nahlédne, že L je CFL: případ $w \notin a^*b^*c^*$ je popsatelný pomocí regulárního jazyka – ty jsou uzavřeny vzhledem ke doplňku; případy typu, kdy $w \in a^*b^*c^*$, ale počet symbolů a je různý od počtu symbolů b atd jsou popsatelné pomocí CFL/PDA. Díky uzavřenosti CFL na sjednocení dostáváme, že L je CFL. L však nemůže být DCFL: kdyby tomu tak bylo, pak by (díky uzavřenosti na doplněk) musel být DCFL i jazyk $\neg L = \{a^nb^nc^n \mid n \geq 1\}$, který však není ani CFL.

Podobně lze ukázat, že například $\neg\{ww \mid w \in \{a,b\}^*\}$ je CFL (najděte pro něj CFG), který však není DCFL.

Důsledek 3.86. *Třída DCFL tvoří vlastní podtřídu třídy bezkontextových jazyků.*

Důkaz. Viz $L = \neg\{a^nb^nc^n \mid n \geq 1\}$ z právě uvedeného příkladu 3.85. □

Kapitola 4

Turingovy stroje a jazyky typu 0

Cílem této kapitoly je definovat a prozkoumat vlastnosti nejsilnějšího z doposud uvažovaných výpočetních modelů — Turingova stroje. Je pojmenován podle matematika Alana Turinga, který ho v roce 1936 definoval. Turingův stroj dokáže realizovat libovolný proces, který lze intuitivně nazvat algoritmem. Ve skutečnosti, jak ukážeme později, je smysluplné definovat pojem algoritmicky řešitelný právě jako řešitelný Turingovým strojem.

4.1 Turingův stroj: model a jeho definice

Neformální popis

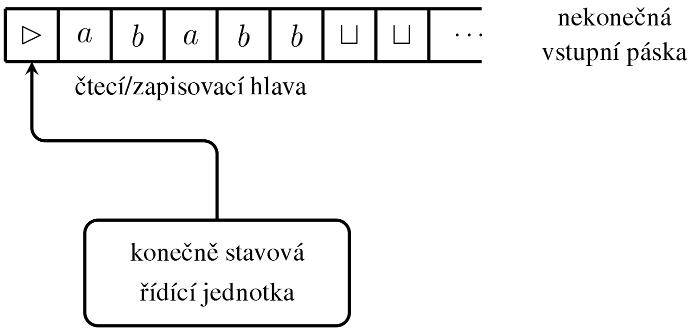
Turingův stroj byl navržen dlouho předtím, než se objevil první počítač. Motivací pro jeho definici byla snaha přesně rozlišit co je a co není vyčíslitelné, tj. co lze a co nelze efektivně vypočítat. Z toho vyplynuly základní požadavky: zaprvé, každý výpočet se musí dát reprezentovat konečným způsobem. Zadruhé, výpočet se má skládat z diskrétních kroků, přičemž každý z nich je mechanicky realizovatelný.

Turingův stroj (anglicky Turing machine, zkráceně TM) má konečnou množinu stavů Q, pásku, která je rozdělena na jednotlivá políčka, a hlavu, která se může po pásce pohybovat doleva a doprava, číst a zapisovat symboly. Na každém políčku pásky je zapsán právě jeden z konečně mnoha páskových (pracovních) symbolů.

Páska je jednosměrně nekonečná. Na nejlevějším (nultém) políčku je zapsán speciální symbol $\triangleright$, označující levý konec pásky. Na začátku výpočtu je na prvním až n-tém, $n \geq 0$, políčku pásky zapsán vstupní řetěz (vstupem tedy může být i prázdný řetěz). Ostatních nekonečně mnoho políček napravo od vstupu je prázdných — tuto skutečnost vyjádříme pomocí speciálního znaku $\sqcup$.

Výpočet začíná v počátečním stavu q_0, přičemž hlava snímá nulté políčko obsahující levou koncovou značku $\triangleright$. Krok výpočtu spočívá v tom, že stroj v závislosti na momentálním stavu a symbolu snímaném hlavou

1. změní svůj stav (či přesněji může změnit),
2. zapíše symbol na políčko snímané hlavou (čímž přepíše symbol, který tam byl zapsán předtím) a
3. posune hlavu o jedno políčko doprava, nebo doleva.



Obrázek 4.1: Turingův stroj

Způsob, jakým se má změnit stav, přepsat symbol a posunout hlava, předepisuje přechodová funkce δ. Stroj *akceptuje* vstupní řetěz, právě když přejde do speciálního akceptujícího stavu q_{accept}. Stroj *zamítá* právě když přejde do speciálního zamítajícího stavu q_{reject}. Na některých vstupech může výpočet běžet nekonečně dlouho, aniž by stroj vstupní slovo akceptoval, nebo zamítnul. V takovém případě říkáme, že stroj pro daný vstup *cyklí*.

Formální definice Turingova stroje

Formálně, Turingův stroj (TM) je 9-tice $\mathcal{M} = (Q, \Sigma, \Gamma, \rhd, \sqcup, \delta, q_0, q_{accept}, q_{reject})$ kde:

- Q je konečná množina *stavů*;
- Σ je konečná množina *vstupních symbolů*;
- Γ je konečná množina *páskových (pracovních) symbolů*, obsahující jakou svou podmnožinu abecedu Σ;
- $\rhd \in \Gamma \setminus \Sigma$ je *levá koncová značka*;
- $\sqcup \in \Gamma \setminus \Sigma$ je symbol označující *prázdné políčko*;
- $\delta : (Q \setminus \{q_{accept}, q_{reject}\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ je *totální přechodová funkce*;
- $q_0 \in Q$ je *počáteční stav*;
- $q_{accept} \in Q$ je *akceptující stav*;
- $q_{reject} \in Q$ je *zamítající stav*;

Navíc požadujeme, aby Turingův stroj nikdy nepřepsal levou koncovou značku jiným symbolem a aby nikdy neposunul svou hlavu vlevo od políčka obsahujícího levou koncovou značku. Formálně, požadujeme aby pro každé $q \in Q$ existoval stav $p \in Q$ takový, že

$$\delta(q, \rhd) = (p, \rhd, R).$$

Množina stavů spolu s přechodovou funkcí se někdy souhrnně označuje jako *řídící jednotka* Turingova stroje.

Konfigurace a výpočet

Abychom mohli přesně definovat pojem výpočtu, potřebujeme, podobně jako u zásobníkových resp. konečných automatů, definovat nejdříve pojem konfigurace. Konfigurace obsahuje kompletní informaci o momentálním stavu výpočtu Turingova stroje, tj. o stavu řídící jednotky, o poloze hlavy na pásce a o obsahu pásky. V libovolném okamžiku výpočtu je na pásce zapsán řetěz tvaru $y\sqcup^\omega$, kde $y \in \Gamma^*$ (má konečnou délku!) a symbol $\sqcup^\omega$ značí nekonečný řetěz

$$\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\sqcup\ldots$$

I když páska Turingova stroje je nekonečná, dokážeme vždy její obsah jednoznačně popsat konečným řetězem: stačí totiž specifikovat obsah neprázdných políček (a těch je konečně mnoho).

Formálně, *konfigurace* je prvek množiny $Q \times \{y\sqcup^\omega \mid y \in \Gamma^*\} \times \mathbb{N}_0$. Konfigurace (q, z, n) specifikuje, že Turingův stroj je ve stavu q, obsah pásky je z a hlava snímá n-té políčko zleva, $n \geq 0$.

Počáteční konfigurace stroje $\mathcal{M}$ pro vstup $w \in \Sigma^*$ je konfigurace

$$(q_0, \rhd w\sqcup^\omega, 0).$$

Akceptující konfigurace je každá konfigurace tvaru

$$(q_{accept}, z, n),$$

kde $z \in \Gamma^*, n \in \mathbb{N}_0$. Podobně *zamítající konfigurace* je každá konfigurace tvaru

$$(q_{reject}, z, n).$$

Na množině všech konfigurací Turingova stroje $\mathcal{M}$ definujeme binární relaci *krok výpočtu*, označujeme $\vdash_{\mathcal{M}}$. Pro libovolný řetěz z (i nekonečný) nad abecedou Γ, nechť z_n označuje n-tý symbol řetězu z (z_0 označuje nejlevější symbol řetězu z). Dále nechť $s_b^n(z)$ označuje řetěz, který získáme tak, že v z nahradíme symbol z_n symbolem b. Pak relace $\vdash_{\mathcal{M}}$ je definována předpisem

$$(p, z, n) \vdash_{\mathcal{M}} \begin{cases} (q, s_b^n(z), n+1) & \text{pro } \delta(p, z_n) = (q, b, R) \\ (q, s_b^n(z), n-1) & \text{pro } \delta(p, z_n) = (q, b, L). \end{cases}$$

Analogicky jako v 3.37 definujeme $\vdash^*_{\mathcal{M}}$ a $\vdash^k_{\mathcal{M}}$, resp. $\vdash^*$ a $\vdash^k$.

Výpočet Turingova stroje $\mathcal{M}$ na vstupu w je posloupnost konfigurací $K_0, K_1, K_2 \ldots$ taková, že

- K_0 je počáteční konfigurace stroje $\mathcal{M}$ pro vstup w a
- $K_i \vdash_{\mathcal{M}} K_{i+1}$ pro všechna $i \geq 0$.

Výpočet může být konečný ale i nekonečný.

Stroj $\mathcal{M}$ *akceptuje* vstupní řetěz $w \in \Sigma^*$ právě když výpočet $\mathcal{M}$ na w je konečný a poslední konfigurace výpočtu je akceptující, tj. právě když

$$(q_0, \rhd w\sqcup^\omega, 0) \vdash^*_{\mathcal{M}} (q_{accept}, z, n).$$

Stroj $\mathcal{M}$ *zamítá* vstupní řetěz $w \in \Sigma^*$ právě když výpočet $\mathcal{M}$ na w je konečný a poslední konfigurace výpočtu je zamítající, tj. právě když

$$(q_0, \rhd w\sqcup^\omega, 0) \vdash^*_{\mathcal{M}} (q_{reject}, z, n).$$

Stroj se *zastaví* na vstupu w právě když výpočet $\mathcal{M}$ na w je konečný, tj. právě když $\mathcal{M}$ akceptuje anebo zamítá w. Samozřejmě, může nastat situace, kdy výpočet $\mathcal{M}$ na w je nekonečný, tj. stroj vstupní řetěz ani neakceptuje, ani nezamítá. V takovém případě říkáme, že stroj pro vstup w *cyklí*.

Turingův stroj se nazývá *úplný*, právě když se pro každý vstup zastaví.

Jazyk akceptovaný Turingovým strojem $\mathcal{M}$ označujeme $L(\mathcal{M})$ a definujeme jej jako množinu řetězů, které $\mathcal{M}$ akceptuje, tj.

$$L(\mathcal{M}) = \{w \in \Sigma^* \mid \mathcal{M} \text{ akceptuje } w\}.$$

Speciálně, jestliže $\mathcal{M}$ je úplný, pak říkáme, že jazyk $L(\mathcal{M})$ je *rozhodovaný* strojem $\mathcal{M}$ nebo že stroj $\mathcal{M}$ *rozhoduje jazyk* $L(\mathcal{M})$.

Příklad 4.1. *Navrhněme Turingův stroj rozhodující jazyk $L = \{a^n b^n c^n \mid n \geq 0\}$, který není bezkontextový. Stroj nejdříve posouvá svou hlavu až na konec vstupního řetězu a kontroluje, zda řetěz zapsaný na pásce je tvaru $a^* b^* c^*$. Přitom vůbec nemění obsah pásky (formálně: zapíše vždy ten symbol, který přečetl). Poté, co hlava přečte první prázdné políčko (formálně: políčko obsahující symbol $\sqcup$), začne se posouvat doleva až na levý konec pásky. Následuje cyklus, ve kterém hlava „vymaže“ (přepíše symbolem X) jeden symbol a, jeden b, jeden c a vrátí se na levý konec pásky. Pokud vstupní řetěz patří do jazyka L, stroj nakonec vymaže na pásce všechny symboly a, b, c a akceptuje. V opačném případě vstup zamítne.*

Nechť $\mathcal{M} = (Q, \Sigma, \Gamma, \rhd, \sqcup, \delta, q_0, q_a, q_r)$, *kde*

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_a, q_r\}$,

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

$\Gamma = \Sigma \cup \{\rhd, \sqcup, X\}$.

Přechodová funkce je určena touto tabulkou:

	$\rhd$	a	b	c	$\sqcup$	X
q_0	$(q_0, \rhd, R)$	(q_0, a, R)	(q_1, b, R)	(q_2, c, R)	$(q_3, \sqcup, L)$	$-$
q_1	$-$	$(q_r, -, -)$	(q_1, b, R)	(q_2, c, R)	$(q_3, \sqcup, L)$	$-$
q_2	$-$	$(q_r, -, -)$	$(q_r, -, -)$	(q_2, c, R)	$(q_3, \sqcup, L)$	$-$
q_3	$(q_4, \rhd, R)$	(q_3, a, L)	(q_3, b, L)	(q_3, c, L)	$(q_3, \sqcup, L)$	(q_3, X, L)
q_4	$-$	(q_5, X, R)	$(q_r, -, -)$	$(q_r, -, -)$	$(q_a, -, -)$	(q_4, X, R)
q_5	$-$	(q_5, a, R)	(q_6, X, R)	$(q_r, -, -)$	$(q_r, -, -)$	(q_5, X, R)
q_6	$-$	$-$	(q_6, b, R)	(q_3, X, L)	$(q_r, -, -)$	(q_6, X, R)

Symbol "–" v tabulce vyjadřuje, že není podstatné, co se zapíše na uvedené místo (můžeme tam doplnit cokoli vyhovující definici). Výpočet stroje $\mathcal{M}$ na vstupu $a^2 b^2 c^2$ je

$$
\begin{array}{llll}
(q_0, \rhd aabbcc\sqcup^\omega, 0) & \vdash\!\!\frac{1}{}\, (q_0, \rhd aabbcc\sqcup^\omega, 1) & \vdash\!\!\frac{1}{}\, (q_0, \rhd aabbcc\sqcup^\omega, 2) & \vdash\!\!\frac{1}{} \\
(q_0, \rhd aabbcc\sqcup^\omega, 3) & \vdash\!\!\frac{1}{}\, (q_1, \rhd aabbcc\sqcup^\omega, 4) & \vdash\!\!\frac{3}{}\, (q_2, \rhd aabbcc\sqcup^\omega, 7) & \vdash\!\!\frac{1}{} \\
(q_3, \rhd aabbcc\sqcup^\omega, 6) & \vdash\!\!\frac{7}{}\, (q_4, \rhd aabbcc\sqcup^\omega, 1) & \vdash\!\!\frac{1}{}\, (q_5, \rhd Xabbcc\sqcup^\omega, 2) & \vdash\!\!\frac{2}{} \\
(q_6, \rhd XaXbcc\sqcup^\omega, 4) & \vdash\!\!\frac{2}{}\, (q_3, \rhd XaXbXc\sqcup^\omega, 4) & \vdash\!\!\frac{5}{}\, (q_4, \rhd XaXbXc\sqcup^\omega, 1) & \vdash\!\!\frac{2}{} \\
(q_5, \rhd XXXbXc\sqcup^\omega, 3) & \vdash\!\!\frac{2}{}\, (q_6, \rhd XXXXXc\sqcup^\omega, 5) & \vdash\!\!\frac{2}{}\, (q_3, \rhd XXXXXX\sqcup^\omega, 5) & \vdash\!\!\frac{6}{} \\
(q_4, \rhd XXXXXX\sqcup^\omega, 1) & \vdash\!\!\frac{6}{}\, (q_4, \rhd XXXXXX\sqcup^\omega, 7) & \vdash\!\!\frac{1}{}\, (q_a, \rhd XXXXXX\sqcup^\omega, 7) &
\end{array}
$$

Rekursivní a rekursivně spočetné jazyky

Jazyk $L \subseteq \Sigma^*$ nazýváme

rekursivně spočetný (Recursively enumerable, *RE*) právě když $L = L(\mathcal{M})$ pro nějaký Turingův stroj $\mathcal{M}$;

rekursivní (Recursive, *Rec*) právě když $L = L(\mathcal{M})$ pro nějaký *úplný* TM $\mathcal{M}$.

Ke každému rekursivnímu jazyku L existuje Turingův stroj, který ho rozhoduje, tj. výpočet na každém vstupním slovu je konečný. Rekursivně spočetný jazyk musí splňovat slabší podmínku: musí pro něj existovat Turingův stroj, který ho akceptuje, tj. akceptuje každé slovo z L, ale výpočet na slovu nepatřícím do L může být buď zamítající, nebo nekonečný.

Rozhodnutelné, nerozhodnutelné a částečně rozhodnutelné problémy

Problém, kdy se má určit, zda řetěz w má vlastnost P, nazýváme

rozhodnutelný právě když množina všech řetězů majících vlastnost P je rekursivní, tj. existuje úplný Turingův stroj $\mathcal{M}$, který akceptuje každý řetěz mající vlastnost P a zamítne každý řetěz, který tuto vlastnost nemá ($\mathcal{M}$ rozhoduje jazyk obsahující právě všechna ta slova, která mají vlastnost P);

nerozhodnutelný právě když není rozhodnutelný;

částečně rozhodnutelný (semirozhodnutelný) právě když množina všech řetězů majících vlastnost P je rekursivně spočetná, tj. existuje Turingův stroj, který akceptuje každý řetěz mající vlastnost P (a zamítá anebo cyklí pro řetěz nemající vlastnost P).

Namísto „problém určit, zda řetěz w má vlastnost P je rozhodnutelný (částečně rozhodnutelný)“ zkráceně říkáme, že *vlastnost P je rozhodnutelná* resp. že *problém P je rozhodnutelný* (částečně rozhodnutelný).

Ačkoli vlastnost rekursivní resp. rekursivně spočetný vypovídá o množinách, zatímco rozhodnutelnost resp. semirozhodnutelnost je vlastnost problémů, jsou oba pojmy úzce spjaty. Platí mezi nimi tato ekvivalence:

P je rozhodnutelný	$\Longleftrightarrow$	jazyk $\{w \mid w$ má vlastnost $P\}$ je rekursivní
L je rekursivní	$\Longleftrightarrow$	problém „$w \overset{?}{\in} L$“ je rozhodnutelný
P je semirozhodnutelný	$\Longleftrightarrow$	jazyk $\{w \mid w$ má vlastnost $P\}$ je rekursivně spočetný
L je rekursivně spočetný	$\Longleftrightarrow$	problém „$w \overset{?}{\in} L$“ je semirozhodnutelný

Turingovy stroje a funkce

Na Turingovy stroje můžeme nahlížet nejen jako na automaty, které akceptují jazyky, ale i jako na zařízení počítající (vyčíslující) funkce na množině řetězů (slov) nad nějakou abecedou, případně též i na množině přirozených čísel. Pro jednoduchost předpokládejme, že přirozená čísla budeme reprezentovat v unární číselné soustavě, tj. číslo $i \in \mathbb{N}_0$ zapíšeme jako řetěz 0^i (nulu reprezentujeme prázdným řetězem). Uvažujme funkci arity k; jejich k argumentů $i_1, i_2, \ldots, i_k$ můžeme jednoznačně reprezentovat řetězem $0^{i_1}10^{i_2}1\ldots10^{i_k}$.

Turingův stroj $\mathcal{M}$ počítá funkci $f : \mathbb{N}_0^k \to \mathbb{N}_0$ právě když $\mathcal{M}$ akceptuje vstupní řetěz $0^{i_1}10^{i_2}1\ldots10^{i_k}$ a obsah jeho pásky v akceptující konfiguraci je $\triangleright 0^m \sqcup^\omega$, kde $m = f(i_1, i_2, \ldots, i_k)$. Nevylučujeme možnost existence vstupu tvaru $0^{i_1}1\ldots10^{i_k}$, který TM zamítne anebo pro který cyklí, resp. který akceptuje, ale nevypočte žádnou hodnotu (obsah pásky není požadovaného tvaru). Podotýkáme, že jeden Turingův stroj může počítat funkci jedné proměnné, jinou funkci dvou proměnných atd.

Funkce $f : \mathbb{N}_0^k \to \mathbb{N}_0$ se nazývá

částečně rekursivní právě když existuje Turingův stroj počítající funkci f;

rekursivní právě když existuje Turingův stroj počítající funkci f a navíc funkce f je totální.

Hodnota částečně rekursivní funkce nemusí být definována pro všechny vstupní argumenty. Všechny běžné aritmetické funkce, jako například součet, násobení, faktoriál, jsou rekursivní. Přirozeným způsobem lze definici rozšířit i pro funkce nad libovolným jiným definičným oborem a oborem hodnot. Zejména tedy můžeme definovat fuknce typu $\Sigma^* \times \ldots \times \Sigma^* \longrightarrow \Sigma^*$ nad řetězci (slovy) nad nějakou abecedou Σ, které můžeme považovat za (jednosměrné) seznamy znaků abecedy. Takto lze snadno definovat obvyklé funkce zřetězení (slov, seznamů) a další standardní funkce pro práci nad seznamy, jako například *head*, *tail*, *cons*, *append* a další.

4.2 Metody konstrukce Turingových strojů

Turingův stroj můžeme programovat podobně jako počítač. Tím, že určujeme přechodovou funkci Turingova stroje, píšeme pro něj vlastně program. Abychom byli schopni naprogramovat i komplikované Turingovy stroje, uvádíme několik triků resp. konceptuálních pojmů, které mohou tento úkol učinit snadnějším.

Zapamatování v řídící jednotce

Prostřednictvím stavu si Turingův stroj může zapamatovat konečně mnoho různých informací. V takovém případě je vhodné zapisovat (pojmenovat) stav jako uspořádanou dvojici. Hodnota v první složce řídí činnost stroje; ve druhé složce je uložena informace, kterou si stroj potřebuje zapamatovat. Definice Turingova stroje zůstává nezměněna, stav jako uspořádaná dvojice je jenom naše vlastní interpretace (označení).

Příklad 4.2. *Mějme jazyk L slov nad $\{a, b\}$, které začínají a končí stejným symbolem, tj.:*

$$L = \{xux \mid x \in \{a, b\},\ u \in \{a, b\}^*\} \cup \{a, b\}.$$

Turingův stroj $\mathcal{M}$ přečte první symbol vstupního řetězu a zapamatuje si ho ve svém stavu — změní stav na $[q_1, a]$ resp. $[q_1, b]$. Dále posouvá hlavu doprava, dokud nenarazí na prázdné políčko. Posune hlavu o jednu pozici doleva a akceptuje právě když symbol zapsaný na snímaném políčku je shodný se symbolem, který si zapamatoval ve svém stavu.
Formálně, $\mathcal{M} = (Q, \Sigma, \Gamma, \rhd, \sqcup, \delta, s, q_{accept}, q_{reject})$, kde
$Q = \{s, q_{accept}, q_{reject}, [q_1, a], [q_2, a], [q_1, b], [q_2, b]\}$,
$\Sigma = \{a, b\}$,
$\Gamma = \Sigma \cup \{\rhd, \sqcup\}$.
Přechodová funkce je určena tabulkou:

	$\rhd$	a	b	$\sqcup$
s	$(s, \rhd, R)$	$([q_1, a], a, R)$	$([q_1, b], b, R)$	$(q_{reject}, -, -)$
$[q_1, a]$	$-$	$([q_1, a], a, R)$	$([q_1, a], b, R)$	$([q_2, a], \sqcup, L)$
$[q_1, b]$	$-$	$([q_1, b], a, R)$	$([q_1, b], b, R)$	$([q_2, b], \sqcup, L)$
$[q_2, a]$	$-$	$(q_{accept}, -, -)$	$(q_{reject}, -, -)$	$-$
$[q_2, b]$	$-$	$(q_{reject}, -, -)$	$(q_{accept}, -, -)$	$-$

Označování symbolů

Označování symbolů je užitečný trik, který najde uplatnění především v situacích, kdy potřebujeme porovnat jisté skupiny symbolů. Použití techniky ilustruje příklad 4.1. V něm se symbol, který už byl „započten", označil (tj. přepsal symbolem X). Jiný způsob užití této techniky ilustruje následující příklad.

Příklad 4.3. *Nechť $L = \{w \mid w \in \{a\}^*, \ |w| = 2^n, \ n \geq 1\}$. Turingův stroj akceptující jazyk L může postupovat tak, že označí polovinu symbolů na pásce a druhou polovinu nechá neoznačenu. Realizace je taková, že prochází slovo zleva doprava a každý druhý symbol a přepíše symbolem A. Pak posune hlavu na začátek pásky. V dalším průchodu se stroj znovu snaží přepsat polovinu doposud neoznačených symbolů a atd. Pokud je délka vstupního slova mocninou 2, stroj dojde do situace, kdy na pásce je jenom jeden symbol a a akceptuje. V opačném případě nastane někdy během výpočtu situace, ve které se stroj snaží rozdělit na dvě stejné poloviny řetězec liché délky; stroj zamítne.*
Formálně, $\mathcal{M} = (Q, \Sigma, \Gamma, \rhd, \sqcup, \delta, s, q_{accept}, q_{reject})$, kde
$Q = \{s, l, n, k, \overline{k}, q_{accept}, q_{reject}\}$,
$\Sigma = \{a\}$,
$\Gamma = \Sigma \cup \{\rhd, \sqcup, A\}$.
Přechodová funkce je určena tabulkou:

	$\rhd$	a	A	$\sqcup$
s	$(s, \rhd, R)$	(l, a, R)	(s, A, R)	$(n, \sqcup, L)$
l	$-$	(s, A, R)	(l, A, R)	$(k, \sqcup, L)$
n	$(s, \rhd, R)$	(n, a, L)	(n, A, L)	$-$
k	$-$	$(\overline{k}, a, L)$	(k, A, L)	$-$
$\overline{k}$	$(q_{accept}, -, -)$	$(q_{reject}, -, -)$	$(\overline{k}, A, L)$	$-$

Otázka 4.4. *Modifikujte stroj $\mathcal{M}$ tak, aby* rozhodoval *jazyk L.*

Násobné stopy

Představme si, že Turingův stroj má „širokou“ pásku, která je rozdělena na k stop, pro libovolné konečné k. Situace pro $k = 3$ je naznačena na obrázku 4.2. Pracovní abeceda stroje obsahuje k-tice symbolů, jeden znak pro každou stopu. Pro lepší názornost je na obrázku k-tice symbolů zapsána ve svislé poloze a jednotlivé položky jsou odděleny vodorovnou čárou.

<table>
<tr><td rowspan="3">$\triangleright$</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td rowspan="3">$\sqcup$</td><td rowspan="3">$\sqcup$</td><td rowspan="3">$\cdots$</td></tr>
<tr><td>$\sqcup$</td><td>$\sqcup$</td><td>$\sqcup$</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
</table>

Obrázek 4.2: Páska se 3 stopami

Příklad 4.5. *Sestrojme Turingův stroj, který jako vstup dostane přirozené číslo zapsané v binární soustavě. Akceptuje právě když vstup je prvočíslem. Stroj „rozdělí“ svou pásku na tři stopy tak, že svou hlavu posouvá doprava a symbol 1 (0) přepíše symbolem* $[1, \sqcup, \sqcup]$ *(*$[0, \sqcup, \sqcup]$*). Na druhou stopu postupně píše binárně čísla 2, 3, 4,… Pro každé číslo* i *zapsané na druhé stopě testuje, zda* i *dělí vstup. Provádí to tak, že od vstupu opakovaně na třetí stopě odečítá* i*. Na obrázku 4.2 je zachycena situace, kdy TM testuje, zda-li číslo 47 je prvočíslem. Na druhé stopě je zapsáno číslo 5, které už dva krát odečetl od 47 a proto na třetí stopě je zapsáno 37.*

Podprogramy

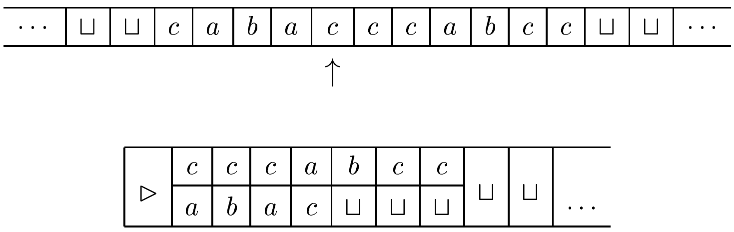
Podobně jako v programování, je i při konstrukci Turingova stroje výhodné použít „modulární“ přístup, tedy přístup shora dolů. Turingův stroj dokáže simulovat libovolný typ procedury, s jakou se můžeme setkat v běžných programovacích jazycích, včetně rekursivních procedur a různých způsobů předávání parametrů. Trik je v tom, že můžeme napsat program Turingova stroje, který budeme využívat jako proceduru. Program má specifikovaný počáteční a koncový stav. Pro koncový stav není zatím definován žádný přechod. Když Turingův stroj chce zavolat tuto proceduru, udělá to tak, že přejde do jejího počátečního stavu. Samozřejmě předpokládáme, že stavy procedury jsou disjunktní se stavy Turingova stroje. Návrat z procedury se realizuje přechodem z koncového stavu procedury do určeného stavu Turingova stroje, který byl například zapsán na pásku jako „návratová adresa“.

4.3 Modifikace Turingových strojů

Jedním z argumentů pro to, aby Turingův stroj byl považován za obecný model výpočtu, je jeho robustnost. Mnohé modifikace TM, které se na první pohled zdají býti silnějšími nebo naopak slabšími, jsou ve skutečnosti výpočtově ekvivalentními, tj. akceptují (i rozhodují) stejnou třídu jazyků.

Turingův stroj s obousměrně nekonečnou páskou

Jedním z nejjednodušších rozšíření Turingova stroje je obousměrně nekonečná páska. Jak vyplývá z jeho názvu, páska je nekonečná jak směrem doprava, tak i doleva. Jeho výpočtová síla je stejná ve srovnání se základním modelem TM. Turingův stroj simulující TM s obousměrně nekonečnou páskou bude mít dvě stopy: jedna bude reprezentovat políčka vpravo od políčka, které se čte na začátku výpočtu (včetně tohoto políčka), druhá bude v obráceném pořadí reprezentovat políčka vlevo od počátečního políčka. Vztah mezi oběma stroji je naznačen na obrázku 4.3. Políčko, na které ukazuje šipka, bylo čteno na začátku výpočtu.



Obrázek 4.3: Simulace obousměrně nekonečné pásky jednosměrně nekonečnou páskou.

Turingův stroj s více páskami

Vícepáskový Turingův stroj se skládá z řídící jednotky, která má k hlav na k (jednosměrně nekonečných) páskách; k je pevně dané přirozené číslo. Na začátku výpočtu je vstupní řetěz zapsán na první pásce. Ostatní pásky mají na nejlevějším políčku zapsán symbol $\triangleright$ (levá koncová značka) a zbylé políčka jsou prázdné (obsahují symbol $\sqcup$). V jednom kroku výpočtu stroj přečte k symbolů zapsaných na políčkách snímaných k hlavami. Na základě této informace, a v závislosti na svém stavu, stroj zapíše nový symbol na každé snímané políčko, změní svůj stav a každou z k hlav buď posune (doprava nebo doleva) anebo nezmění její pozici. Přechodová funkce k-páskového TM je zobrazení množiny $(Q \setminus \{q_{accept}, q_{reject}\}) \times \Gamma^k$ do množiny $Q \times \Gamma^k \times \{L, R, S\}^k$ (symbol S vyjadřuje skutečnost, že pozice hlavy se nemění).

Příklad 4.6. *Sestrojme dvoupáskový Turingův stroj rozhodující jazyk*

$$L = \{a^n b^{n^2} \mid n \geq 0\} .$$

Jazyk L obsahuje slova nad abecedou $\{a, b\}$ taková, že počet symbolů b je druhou mocninou počtu symbolů a.
Výpočet Turingova stroje $\mathcal{M}$ na vstupu w bude probíhat podle tohoto schématu:

1. *Stroj současně posouvá obě hlavy doprava a na druhou pásku zapisuje symbol A, dokud na první pásce nenarazí na první symbol b (stav q). Druhou hlavu posune na začátek pásky (stav n).*
2. *Druhá hlava hledá symbol A (stav h_A); když ho najde, přepíše ho symbolem B (stav n_A) a přesune se na začátek pásky.*

3. Obě hlavy se současně posouvají směrem doprava (stav o), dokud druhá hlava nenarazí na první symbol $\sqcup$.

4. Druhá hlava se posune na začátek pásky (stav n) a výpočet pokračuje bodem 2.

5. K akceptování dojde, když jsou současně *splněny tyto podmínky:*

(a) obě hlavy současně přečetly symbol $\sqcup$ a

(b) všechny symboly A na druhé pásce byly přepsány symbolem B (toto se ověří ve stavu k).

Formálně, $\mathcal{M} = (Q, \Sigma, \Gamma, \rhd, \sqcup, \delta, s, q_{accept}, q_{reject})$, *kde*

$Q = \{q, n, h_A, n_A, o, k, q_{accept}, q_{reject}\}$,

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

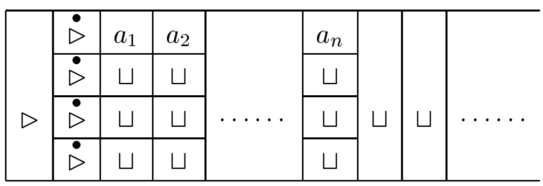
$\Gamma = \Sigma \cup \{\rhd, \sqcup, A, B\}$.

Přechodová funkce je určena takto:

$$
\begin{array}{ll}
\delta(q,\rhd,\rhd) = (q,\rhd,\rhd,R,R) & \delta(n_A,b,B) = (n_A,b,B,S,L) \\
\delta(q,a,\sqcup) = (q,a,A,R,R) & \delta(n_A,b,\rhd) = (o,b,\rhd,S,R) \\
\delta(q,b,\sqcup) = (n,b,\sqcup,S,L) & \\
 & \delta(o,b,B) = (o,b,B,R,R) \\
\delta(n,b,A) = (n,b,A,S,L) & \delta(o,b,A) = (o,b,A,R,R) \\
\delta(n,b,B) = (n,b,B,S,L) & \delta(o,b,\sqcup) = (n,b,\sqcup,S,L) \\
\delta(n,b,\rhd) = (h_A,b,\rhd,S,R) & \delta(o,\sqcup,\sqcup) = (k,\sqcup,\sqcup,S,L) \\
 & \\
\delta(h_A,b,B) = (h_A,b,B,S,R) & \delta(k,\sqcup,B) = (k,\sqcup,B,S,L) \\
\delta(h_A,b,A) = (n_A,b,B,S,L) & \delta(k,\sqcup,\rhd) = (q_{accept},\sqcup,\rhd,S,S)
\end{array}
$$

Pro všechny ostatní kombinace (stav, čtené symboly) definujeme hodnotu funkce δ jako přechod do zamítajícího stavu q_{reject} .

Nechť $\mathcal{K}$ je k-páskový TM. Ukážeme, jak je možno sestrojit (jednopáskový) TM $\mathcal{J}$, který simuluje výpočet stroje $\mathcal{K}$. Páska stroje $\mathcal{J}$ bude rozdělena na k stop. Každá stopa reprezentuje obsah jedné pásky stroje $\mathcal{K}$. Navíc, každá stopa obsahuje právě jeden speciálně označený symbol indikující, které políčko právě snímá odpovídající hlava stroje $\mathcal{K}$. Na začátku výpočtu si $\mathcal{J}$ upraví odpovídajícím způsobem svou pásku. Pro vstup $a_1a_2\ldots a_n$ bude mít upravená páska podobu zachycenou na obrázku 4.4.



Obrázek 4.4: Simulace 4 pásek pomocí jedné.

Aby stroj $\mathcal{J}$ odsimuloval jeden krok stroje $\mathcal{K}$, musí $\mathcal{J}$ postupně přečíst každé políčko označené speciální značkou • a označené symboly si zapamatovat ve svém stavu. Formálně, $\mathcal{J}$ prochází svou pásku zleva doprava, dokud nenajde všech k značek a ve svém stavu si pamatuje uspořádanou k-tici symbolů. Hlava se vrátí na levou koncovou značku. Po shromáždění všech potřebných údajů $\mathcal{J}$ zjistí, který krok by vykonal stroj $\mathcal{K}$ a realizuje ho tím, že znovu prochází svou pásku. Vždy, když narazí na speciální značku, změní patřičným způsobem obsah políčka a posune značku doleva anebo doprava. Nakonec $\mathcal{J}$ opět posune svou hlavu na levou koncovou značku a začíná simulovat další krok stroje $\mathcal{K}$. Na rozdíl od předcházející simulace (obousměrně nekonečná $\longrightarrow$ jednosměrně nekonečná páska), je v tomto případě na simulaci jednoho kroku původního stroje potřebných více kroků.

Otázka 4.7. *Navrhněte TM se třemi páskami, akceptující jazyk z příkladu 4.1. Srovnejte počet kroků, které musí udělat stroj z příkladu 4.1, než akceptuje vstup délky n, a které musí udělat Vámi navržený stroj.*

Nedeterministický Turingův stroj

Rozdíl oproti původnímu (deterministickému) Turingovu stroji spočívá v tom, že pro daný stav a snímaný symbol má nedeterministický stroj obecně několik možností pro následující krok. Každá možnost zahrnuje nový stav, symbol, který se má zapsat na pásku a směr, kterým se má posunout hlava. Nedeterministický stroj akceptuje právě když existuje výpočet (tj. nějaká posloupnost výběru kroků) vedoucí do akceptující konfigurace.

Definice 4.8. *Nedeterministický Turingův stroj* je 9-tice $\mathcal{M} = (Q, \Sigma, \Gamma, \triangleright, \sqcup, \delta, q_0, q_{accept}, q_{reject})$, kde význam všech složek je stejný jako v definici 4.1, s výjimkou přechodové funkce. Ta je definována jako (totální) zobrazení $\delta : (Q \setminus \{q_{accept}, q_{reject}\}) \times \Gamma \to 2^{Q\times\Gamma\times\{L,R\}}$.

Stejně jako v případě deterministických TM, je možné definovat jazyk $\mathcal{M}$ pomocí pojmů konfigurace a krok výpočtu; jediná změna je v definice relace krok výpočtu: relace $\vdash_{\mathcal{M}}$ je definována předpisem

$$(p, z, n) \vdash_{\mathcal{M}} \begin{cases} (q, s_b^n(z), n+1) & \text{právě když } \exists\,(q, b, R) \in \delta(p, z_n) \\ (q, s_b^n(z), n-1) & \text{právě když } \exists\,(q, b, L) \in \delta(p, z_n) \end{cases}$$

Výpočet TM $\mathcal{M}$ na vstupu w si můžeme pro názornost představit jako strom (tzv. *výpočtový strom*), jehož vrcholy jsou konfigurace (kořen je počáteční konfigurace $\mathcal{M}$ na w) a jednotlivé cesty odpovídají různým výpočtům $\mathcal{M}$ na w. Stroj $\mathcal{M}$ akceptuje vstup w právě když ve výpočtovém stromu *existuje* cesta z kořene do listu odpovídajícímu akceptující konfiguraci.

I v tomto případě, i když je to možná poněkud překvapující, se dá navrhnout simulace (deterministickým) Turingovým strojem.

Věta 4.9. *Pro každý nedeterministický Turingův stroj existuje ekvivalentní deterministický Turingův stroj.*

Důkaz. Nechť $\mathcal{N} = (Q, \Sigma, \Gamma, \triangleright, \sqcup, \delta, q_0, q_{accept}, q_{reject})$ je nedeterministický TM. Navrhneme deterministický TM $\mathcal{D}$ simulující nedeterministický stroj $\mathcal{N}$.

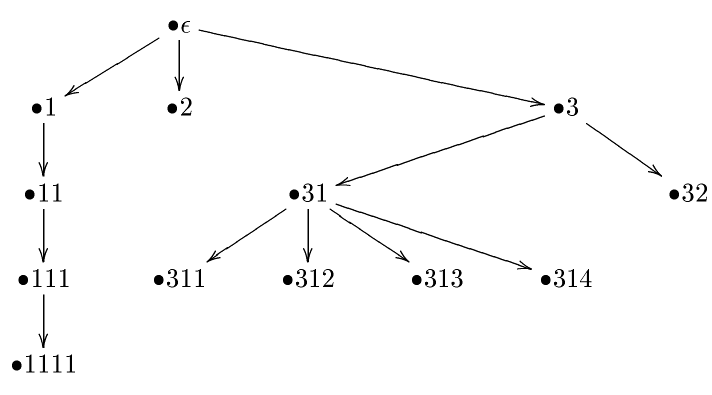
Simulace je založena na tom, že stroj $\mathcal{D}$ prozkoumá všechny výpočty stroje $\mathcal{N}$ a zjistí, jestli některý z nich obsahuje akcetpující konfiguraci. Když si výpočty stroje $\mathcal{N}$ na vstupu w představíme jako výpočtový strom, tak prohledání výpočtů znamená vlastně prohledání stromu. Je třeba si ale uvědomit, že z dvou možných způsobů, které přicházejí do úvahy — prohledávání do hloubky a do šířky — je jen jeden korektní. Protože výpočtový strom může být nekonečný, můžeme se při prohledávání do hloubky dostat do situace, kdy sledujeme jednu konkrétní, nekonečnou cestu a nikdy se nedostaneme k prozkoumání ostatních cest, z nichž některá může vést do akceptující konfigurace.

Stroj $\mathcal{D}$ bude mít 3 pásky. První páska obsahuje vstupní řetěz a její obsah se v průběhu výpočtu nemění. Na druhé pásce simuluje $\mathcal{D}$ výpočet stroje $\mathcal{N}$. Na třetí pásce si $\mathcal{D}$ uchovává informaci určující, který vrchol výpočtového stromu právě prohledává.

Specifikujeme nejdříve, jakým způsobem stroj $\mathcal{D}$ reprezentuje informaci na třetí pásce. Každý vrchol výpočtového stromu má nejvýše b následníků, kde b je maximální kardinalita množiny $\delta(q, a)$,

$$b = \max\{|\delta(q, a)| \mid q \in Q, a \in \Gamma\} .$$

Každý vrchol stromu označíme řetězem nad abecedou $\Sigma_b = \{1, 2, \ldots, b\}$. Například řetězem 314 označíme vrchol (konfiguraci), do kterého se dostaneme z kořenu (počáteční konfigurace) když si v prvním kroku výpočtu vybereme třetího následníka, v druhém kroku prvního a v třetím kroku čtvrtého následníka (obr. 4.5). Každý symbol v řetězu určuje, kterého následníka si máme vybrat při simulaci dalšího kroku výpočtu. Může nastat situace, kdy symbolu neodpovídá žádný následník – v takovém případě je řetěz kódem neplatného výpočtu. Kořen stromu je označen řetězem ϵ. Informace na třetí pásce stroje $\mathcal{D}$ je reprezentována právě jako řetěz nad abecedou Σ_b.



Obrázek 4.5: Výpočtový strom a jeho značení

Teď již jsme připraveni popsat výpočet stroje $\mathcal{D}$.

1. Na začátku obsahuje první páska vstup w; druhá a třetí páska jsou prázdné.
2. Zkopíruje obsah první pásky na druhou pásku.

3. Na druhé pásce simuluje výpočet $\mathcal{N}$ na vstupu w, přičemž v každém kroku se rozhoduje podle následujícího symbolu z třetí pásky, kterým směrem pokračovat v simulaci. Když všechny symboly z třetí pásky byly přečteny, nebo řetěz na třetí pásce je kódem neplatného výpočtu, nebo simulovaný výpočet se dostal do zamítající konfigurace, tak $\mathcal{D}$ pokračuje bodem 4. Když se simulovaný výpočet dostane do akceptující konfigurace, tak $\mathcal{D}$ akceptuje.
4. Řetěz na třetí pásce nahradí řetězem, který za ním následuje v lexikografickém uspořádání. Pokračuje bodem 2. (tj. simulací dalšího výpočtu stroje $\mathcal{N}$).

Je tedy vidět, že jazyky *akceptované* strojem $\mathcal{D}$ a strojem $\mathcal{N}$ jsou si rovny, což jsme měli dokázat. □

Poznamejme, že pokud stroj $\mathcal{N}$ z právě uvedeného důkazu neakceptuje a zamítá, pak simulující stroj $\mathcal{D}$ může cyklit, což však na akceptovaný jazyk nemá vliv.

Důsledek 4.10. *Jazyk je rekursivně spočetný právě když je akceptován nějakým nedeterministickým Turingovým strojem.*

Konečně poznamenejme, že tvrzení analogické větě 4.9 by platilo i pro úplný nedetermistický TM a rozhodování jazyků (definici tohoto stroje ponecháváme čtenáři; zejména by v libovolném výpočetním stromu takového stroje neexistovaly žádné nekonečné cesty). Simulující deterministický stroj $\mathcal{D}$ by pak bylo možno zkonstruovat tak, aby byl rovněž úplný.

Turingův stroj s oddělenou vstupní páskou

Jedná se o model, ve kterém má stroj dvě pásky: vstupní a pracovní. Vstupní řetězec je zapsán na vstupní pásce, z níž může stroj jen číst (nesmí měnit její obsah). V závislosti na tom, zda se hlava na vstupní pásce může pohybovat jenom doprava resp. oběma směry, hovoříme o *on-line* resp. *off-line* Turingových strojích. Zřejmě tato modifikace neovlivní výpočetní sílu TM.

Všechny doposud uvažované modifikace byly rozšířeními základního modelu. Následující modely by na první pohled mohly mít méně výpočetních možností než Turingovy stroje; o všech však prokážeme, že jejich výpočetní síla je stejná jako u Turingových strojů.

Stroj se dvěma zásobníky

Jedná se o Turingův stroj se vstupní páskou, který má místo pracovní pásky dva zásobníky. Výpočet (jednopáskového) TM dokáže stroj se dvěma zásobníky simulovat takto: do prvního zásobníku si uloží tu část pásky TM, která je vlevo od políčka právě snímaného hlavou, přičemž na vrcholu zásobníku je symbol bezprostředně vlevo od snímaného symbolu. Zbývající část pásky si uloží do druhého zásobníku tak, že právě čtený symbol je na vrcholu zásobníku. Pohyb hlavy doleva (doprava) je simulován přesunem vrcholového symbolu prvního (druhého) zásobníku na vrchol druhého (prvního) zásobníku. Například páska

(pozice hlavy je naznačena šipkou)

$$
\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|c|l}
\hline
\triangleright & a & a & c & a & b & a & c & c & c & a & b & \sqcup & \sqcup & \cdots \\
\hline
 & & & & & & \uparrow & & & & & & & &
\end{array}
$$

je simulována zásobníky

$$
\begin{array}{|c|}
b \\ \hline
a \\ \hline
c \\ \hline
a \\ \hline
a \\ \hline
\perp \\ \hline
\end{array}
\qquad\qquad
\begin{array}{|c|}
a \\ \hline
c \\ \hline
c \\ \hline
c \\ \hline
a \\ \hline
b \\ \hline
\perp \\ \hline
\end{array}
$$

Všimněme si ještě, jakým způsobem stroj manipuluje se zásobníkem. V každém kroku zjišťuje, jaký symbol je uložen na vrcholu zásobníku a následně buď tento symbol ze zásobníku odstraní, nebo na vrchol přidá nový symbol.

Stroj s dvěma počitadly

Obecně, k-počitadlový stroj je Turingův stroj se vstupní páskou a s k pracovními páskami, který navíc splňuje tyto požadavky:

- na každou z pracovních pásek může stroj zapisovat jenom jeden ze dvou symbolů: $\triangleright$ (levá koncová značka) a $\sqcup$ (symbol pro prázdné políčko),
- symbol $\triangleright$ je na začátku zapsán na levém krajním políčku pracovní pásky a nikdy se nesmí objevit na jiném políčku.

Když hlava snímá i-té políčko pracovní pásky, můžeme to interpretovat jako fakt, že stroj má uloženo v paměti celé nezáporné číslo i. Přitom v každém kroku výpočtu stroj testuje, zda číslo uložené na pásce je rovno nule (čte se symbol $\triangleright$) nebo různé od 0 (čte se symbol $\sqcup$). V každém kroku může stroj hodnotu zapamatovaného čísla zvětšit anebo zmenšit (to v případě, že je nenulové) o jedničku tím, že svou hlavu posune doprava nebo doleva. Namísto termínu počitadlo se lze často setkat s termínem čítač (anglicky counter).

Mírná modifikace (rozdíl se týká jen způsobu zápisu) počitadlového stroje je známá jako *Minského stroj* (zavedl matematik Marvin Minsky). Formálně lze tento stroj (s k počitadly) definovat jako konečnou posloupnost instrukcí s návěštími (program) tvaru:

$$l_0 : \textit{příkaz}_0, \ldots, l_{n-1} : \textit{příkaz}_{n-1}, l_n : \text{stop},$$

kde každá z instrukcí $l_I : \textit{příkaz}_i,\ i \in \langle 0, n-1 \rangle$ je tvaru buď

$$
\begin{array}{lll}
l_p : & c_i := c_i + 1;\ \text{goto}\ l_q & 1 \le i \le k, \quad \text{nebo} \\
l_p : & \text{if}\ c_i = 0\ \text{then goto}\ l_q & \\
 & \quad \text{else}\ c_i := c_i - 1;\ \text{goto}\ l_r & 1 \le i \le k\ .
\end{array}
$$

Počáteční konfigurace , pokud není řečeno jinak, je definována tak, že čítač instrukcí je nastaven na l_0, hodnoty počitadel $c_1, \ldots, c_k$ kódují požadovaný vstup.

V literatuře je možno se setkat i s dalšími modifikacemi. Všechny mají společné to, že stroj si může do paměti uložit (několik) libovolně velkých čísel, přičemž o každém z nich je schopen zjistit jenom to, jestli je rovno 0 anebo různé od 0. Navíc, v jednom kroku výpočtu může změnit hodnotu každého z čísel maximálně o 1, či jinou, předem danou celočíselnou konstantu.

Nás bude zajímat vztah počitadlových a Turingovych strojů. Nejdříve ukážeme, že dvě počitadla se dají simulovat jedním zásobníkem. Jak už víme, Turingův stroj se dá simulovat strojem s dvěma zásobníky. Spojení obou poznatků nám umožní dokázat ekvivalenci Turingovych strojů a strojů se 4 počitadly.

Lemma 4.11. *Stroj s jedním zásobníkem se dá simulovat strojem s dvěma počitadly.*

Důkaz. Nechť $\mathcal{M}$ je stroj s jedním zásobníkem, jehož pracovní abeceda je tvořena symboly $Z_0, \ldots, Z_{k-1}$ (tj. stroj má k pracovních symbolů). Předpokládejme, že obsah zásobníku stroje $\mathcal{M}$ je $Z_{i_1} \cdots Z_{i_m}$ (vrchol zásobníku je vpravo). Chceme ukázat, jakým způsobem je možné tutéž informaci uložit do jednoho počitadla. K tomu si stačí uvědomit, že na obsah zásobníku, tj. řetěz $Z_{i_1} \cdots Z_{i_m}$ můžeme nahlížet jako na poziční zápis čísla $j = i_1 \cdot k^{m-1} + i_2 \cdot k^{m-2} + \cdots + i_{m-1} \cdot k + i_m$ v k-adické soustavě. Na druhé straně, v počitadle si stroj pamatuje číslo zapsané v unární soustavě. Proto zásobník $Z_{i_1} \cdots Z_{i_m}$ se dá jednoznačně reprezentovat v počitadle jako číslo $j = i_1 \cdot k^{m-1} + i_2 \cdot k^{m-2} + \cdots + i_{m-1} \cdot k + i_m$.

Dále potřebujeme ukázat, jak stroj s počitadly dokáže simulovat operace nad zásobníkem, tj. přidání a odebrání symbolu ze zásobníku a přečtení symbolu z vrcholu zásobníku (srovnej s přecházejícím odstavcem).

- **Přidání symbolu do zásobníku** Předpokládejme, že na vrchol zásobníku je přidán symbol Z_r. Pak číslo reprezentující tento nový obsah zásobníku je právě $j \cdot k + r$. To znamená, že potřebujeme vynásobit předešlý obsah počitadla číslem k a připočíst k němu r. Proto opakovaně (předpokládáme, že číslo j je uloženo v 1. počitadle a 2. počitadlo je prázdné)
  - 1. počitadlo posune svou hlavu o 1 políčko doleva zatímco
  - 2. počitadlo posune svou hlavu o k políček doprava.

  Jakmile 1. počitadlo je na dně (čte se symbol $\triangleright$), pak 2. počitadlo obsahuje číslo $k \cdot j$, k němuž lehce (pomocí konečně stavové řídící jednotky, bez manipulace s počitadly) připočteme r ($r \in \langle 0, k-1 \rangle$).
- **Odebrání symbolu ze zásobníku** Z vrcholu je odebrán zymbol Z_{i_m} a číslo reprezentující změněný zásobník je j div k. Abychom dosáhli odpovídající situaci v počitadle, opakovaně
  - 1. počitadlo posune svou hlavu o k políček doleva a
  - 2. počitadlo posune svou hlavu o 1 políčko doprava.

  Jakmile 1. počitadlo dosáhne dna, je ve 2. počitadle číslo j div k.
- **Přečtení symbolu na vrcholu zásobníku** V 1. počitadle je uloženo číslo j a na zjištění, který symbol je na vrcholu zásobníku musíme vypočítat j mod k. Toho dosáhneme tak, že kopírujeme obsah 1. počitadla do 2. počitadla a ve stavové jednotce přitom počítáme j mod k.

$\square$

Důsledek 4.12. *Každý Turingův stroj lze simulovat strojem se 4 počitadly.*

Nyní ukážeme, že právě navrženou simulaci je možné ještě zoptimalizovat.

Věta 4.13. *Každý Turingův stroj lze simulovat strojem se 2 počitadly.*

Důkaz. Potřebujeme ukázat, že stroj s 4 počitadly se dá simulovat strojem s 2 počitadly. Nechť 4 simulovaná počitadla mají pořadě hodnoty i, j, k, l. Jedno simulující počitadlo může reprezentovat všechny 4 výše uvedená počitadla číslem $n = 2^i 3^j 5^k 7^l$ (2, 3, 5, 7 jsou prvočísla, a tedy z n lze jednoznačně určit hodnoty i, j, k, l). Nyní zvýšit číslo i, j, k nebo l o 1 znamená násobit číslem 2, 3, 5 nebo 7. K tomu využijeme 2. počitadlo, které nastavíme na nulu a opakovaně

- 1. počitadlo posune svou hlavu o 1 políčko doleva zatímco
- 2. počitadlo posune svou hlavu o 2 (resp. 3, resp. 5, resp. 7) políček doprava.

Jakmile 1. počitadlo je na nule, 2. počitadlo obsahuje číslo $2n$ (resp. $3n$, resp. $5n$, resp. $7n$).

Analogicky snížit i, j, k nebo l o 1 znamená dělit číslem 2, 3, 5 nebo 7.

K dokončení zbývá ukázat, jak se provede test na nulu (tzn. jak se zjistí, zda obsah konkrétního počítadla je roven nule nebo různý od nuly). K tomu se obsah 1. počitadla (n) zkopíruje do 2. počitadla a přitom se testuje, zda n mod 2 (resp. n mod 3, resp. n mod 5, resp. n mod 7) je rovno nule. □

Důsledek 4.14. *Libovolný stroj s k počitadly lze simulovat strojem se 2 počitadly.*

Upozorněme, že simulace jednoho kroku TM si vyžaduje obrovský počet kroků stroje se 2 počitadly. Stroj s jedním počitadlem je slabší než TM – de facto se jedná o speciální případ PDA se zásobníkovou abecedou $\{Z_0, I\}$, kde Z_0 smí označovat pouze dno zásobníku (takto je umožněn test na nulu) a počet symbolů I na zásobníku odpovídá hodnotě počitadla. Jazyky akceptované těmito stroji s jedním čítačem se v anglické literatuře nazývají *one-counter languages*. Konečně upozorněme, že pro vlastní převod libovolného vstupního slova Turingova stroje do počáteční konfigurace stroje s počitadly potřebujeme 3 počitadla; konstrukci zde neuvádíme a lze ji nalézt v literatuře.

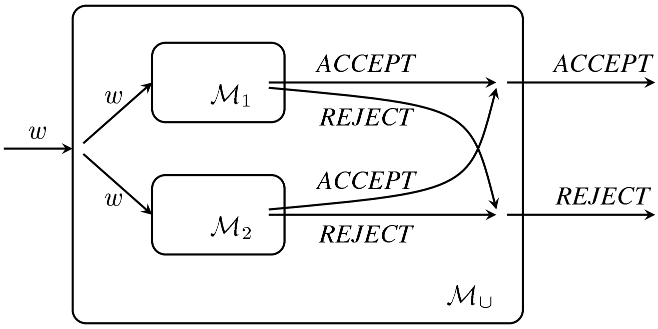
4.4 Vlastnosti rekursivních a rekursivně spočetných jazyků

Cílem je prozkoumat uzavřenost vůči elementárním operacím $\cup, \cap, \cdot, ^*$ a komplementu. V případě prvních čtyř budou důkazové techniky podobné těm, které byly použity při zkoumání uzávěrových vlastností regulárních a bezkontextových jazyků.

Věta 4.15. *Třídy rekursivních a rekursivně spočetných jazyků jsou uzavřeny vzhledem k operacím $\cup$, $\cap$, $\cdot$, a *.*

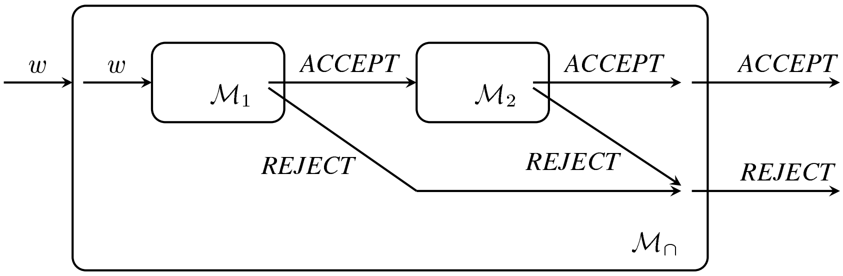
Důkaz. Nechť L_1, L_2 jsou jazyky akceptovány Turingovými stroji $\mathcal{M}_1, \mathcal{M}_2$. Bez újmy na obecnosti můžeme předpokládat, že $\mathcal{M}_1$ a $\mathcal{M}_2$ mají disjunktní množiny stavů.

Nedeterministický stroj $\mathcal{M}_\cup$, akceptující $L_1 \cup L_2$, dostaneme sjednocením strojů $\mathcal{M}_1$ a $\mathcal{M}_2$ (obr. 4.6). Stroj $\mathcal{M}_\cup$ bude mít navíc nový počáteční stav. V prvním kroku výpočtu $\mathcal{M}_\cup$ nedeterministicky přejde buď do počátečního stavu stroje $\mathcal{M}_1$, nebo do počátečního stavu stroje $\mathcal{M}_2$. V dalším simuluje výpočet zvoleného stroje.



Obrázek 4.6: Konstrukce TM pro sjednocení dvou jazyků.

Stroj $\mathcal{M}_\cap$, akceptující $L_1 \cap L_2$ (obr. 4.7), bude mít pásku se třemi stopami. Na první stopě je zapsán vstupní řetěz a její obsah se v průběhu výpočtu nemění. Stroj $\mathcal{M}_\cap$ okopíruje svůj vstup w na druhou stopu a na ní simuluje výpočet $\mathcal{M}_1$ na w. V případě, že $\mathcal{M}_1$ akceptoval, okopíruje vstup w na třetí stopu a na ní pak simuluje výpočet $\mathcal{M}_2$ na w. Jestliže $\mathcal{M}_2$ akceptoval, pak $\mathcal{M}_\cap$ též akceptuje.



Obrázek 4.7: Konstrukce TM pro průnik dvou jazyků.

Nedeterministický stroj $\mathcal{M}_\circ$, akceptující $L_1 \cdot L_2$, bude mít pásku se třemi stopami. Na první stopě je zapsáno vstupní slovo. Stroj překopíruje nějaký (může být i prázdný) prefix vstupního slova na druhou stopu — délku prefixu určí nedeterministicky. Symboly, které byly okopírovány se na první stopě označkují. Na druhé stopě pak $\mathcal{M}_\circ$ simuluje výpočet $\mathcal{M}_1$. V případě, že simulovaný výpočet skončí v akceptující konfiguraci, tak $\mathcal{M}_\circ$ překopíruje na třetí stopu zbylou (neoznačkovanou) část vstupu a simuluje výpočet $\mathcal{M}_2$ na řetězu zapsaném na třetí stopě. $\mathcal{M}_\circ$ akceptuje právě když $\mathcal{M}_2$ akceptuje.

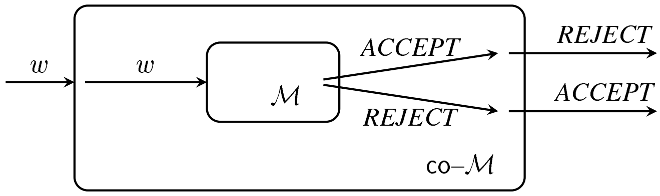
Nedeterministický stroj $\mathcal{M}_*$, akceptující L_1^* je zobecněním stroje $\mathcal{M}_\circ$.

Za předpokladu, že stroje $\mathcal{M}_1$ a $\mathcal{M}_2$ jsou úplné, budou i stroje $\mathcal{M}_\cup, \mathcal{M}_\cap, \mathcal{M}_\cdot$ a $\mathcal{M}_*$ úplné. To dokazuje platnost věty i v případě rekursivních jazyků. $\square$

Poslední elementární operací nad jazyky je komplement (doplněk). Vzhledem k této operaci se třídy rekursivních a rekursivně spočetných jazyků chovají rozdílně. Zatímco komplement rekursivního jazyka je vždy rekursivní jazyk, komplement rekursivně spočetného jazyka nemusí být rekursivně spočetný.

Věta 4.16. *Třída rekursivních jazyků je uzavřena vzhledem k operaci komplementu.*

Důkaz. Nechť L je jazyk akceptovaný úplným deterministickým Turingovym strojem $\mathcal{M} = (Q, \Sigma, \Gamma, \rhd, \sqcup, \delta, q_0, t, r)$. Stroj co–$\mathcal{M}$, akceptující jazyk co–$L = \Sigma^* - L$, získáme tak, že všude v definici přechodové funkce δ zaměníme navzájem akceptující stav q_{accept} a zamítající stav q_{reject} (obr. 4.8). Protože $\mathcal{M}$ je úplný, bude i co–$\mathcal{M}$ úplný. □



Obrázek 4.8: Konstrukce TM pro komplement rekursivního jazyka.

Poznamenejme, že pokud stroj $\mathcal{M}$ z předchozího důkazu není úplný, pak jazyk L(co–$\mathcal{M}$) nemusí být roven co–L. Stačí uvážit slovo w, na kterém $\mathcal{M}$ cyklí. Pak i co–$\mathcal{M}$ na w cyklí, a tedy $w \notin L$(co–$\mathcal{M}$). Současně však $w \in$ co–L. Z uvedené úvahy samozřejmě ještě nevyplývá, že třída rekursivně spočetných jazyků není uzavřena vůči komplementu.

Věta 4.17. *Nechť jazyk L i jeho komplement co–L jsou rekursivně spočetné. Pak jazyky L a co–L jsou rekursivní.*

Důkaz. Předpokládejme, že $\mathcal{M}_1$ a $\mathcal{M}_2$ jsou Turingovy stroje akceptující pořadě jazyky L a co–L. Sestrojíme Turingův stroj $\mathcal{M}$, který bude na vstupu w současně simulovat výpočet $\mathcal{M}_1$ na w i výpočet $\mathcal{M}_2$ na w. Formálně, stroj $\mathcal{M}$ bude mít dvě stopy, jednu pro každý simulovaný výpočet. $\mathcal{M}$ střídavě simuluje jeden krok výpočtu stroje $\mathcal{M}_1$ (na první stopě) a jeden krok výpočtu stroje $\mathcal{M}_2$ (na druhé stopě). $\mathcal{M}$ akceptuje vstup, právě když ho akceptuje $\mathcal{M}_1$ a $\mathcal{M}$ zamítá vstup, právě když ho akceptuje $\mathcal{M}_2$. Protože každý řetěz w patří buď do jazyka L anebo do jazyka co–L, výpočet $\mathcal{M}$ se pro každý vstup zastaví. Proto L je rekursivní. Rekursivita jazyka co–L plyne z věty 4.16. □

Poznamejme, že výše uvedená věta je známa jako Postova věta a je též uváděna jako tvrzení: *L je rekursivní $\iff$ L a co–L jsou rekursivně spočetné.* Platnost tohoto tvrzení je vzhledem k právě dokázaným větám 4.17 a 4.16 zřejmá.

Důsledkem uzavřenosti třídy rekursivně spočetných jazyků k operaci komplementu by byla rovnost tříd rekursivních a rekursivně spočetných jazyků. V následující kapitole (věta 5.6) ukážeme, že tyto dvě třídy nejsou stejné, a proto třída rekursivně spočetných jazyků *není* uzavřena na komplement.

Věty 4.16 a 4.17 mají i další zajímavé důsledky. Například, že pro jazyk L a jeho komplement co–L může nastat jen jedna z těchto tří možností:

1. oba jazyky L a co–L jsou rekursivní;
2. žádný z jazyků L a co–L není rekursivně spočetný a

3. jeden z jazyků L a co–L je rekursivně spočetný ale není rekursivní, druhý není rekursivně spočetný.

4.5 Turingovy stroje a jazyky typu 0

Cílem této části je ukázat, že gramatiky typu 0 (též známé jako frázové gramatiky) jsou výpočetně ekvivalentní Turingovým strojům. Jinými slovy, že třída jazyků generovaných gramatikami typu 0 je právě třída rekursivně spočetných jazyků. I když oba formalismy jsou stejně expresivní, rozdíl mezi nimi je ve způsobu, jakým popisují uvedenou třídu jazyků. Zatímco TM je ve své podstatě návodem, *jak rozpoznat*, zda dané slovo patří do jazyka, tak formální gramatika je návodem, *jak vytvořit* slovo patřící do jazyka.

První z následujících dvou lemmat prokazuje, že každá gramatika typu 0 generuje rekursivně spočetný jazyk; druhá pak opačnou implikaci.

Lemma 4.18. *Nechť L je jazyk generovaný gramatikou typu 0. Pak L je rekursivně spočetný.*

Důkaz. Nechť L je jazyk generovaný gramatikou $\mathcal{G} = (N, \Sigma, P, S)$ typu 0. Sestrojíme nedeterministický Turingův stroj $\mathcal{M}$ akceptující jazyk L. Stroj $\mathcal{M}$ bude mít dvě stopy. Na první stopě je zapsán vstupní řetěz a její obsah se v průběhu výpočtu nemění. Na druhé stopě simuluje $\mathcal{M}$ odvození v $\mathcal{G}$ a je na ní zapsána (do daného okamžiku vygenerovaná) větná forma α gramatiky $\mathcal{G}$. $\mathcal{M}$ inicializuje obsah druhé stopy na S a pak $\mathcal{M}$ opakovaně provádí tyto kroky:

1. Nedeterministicky vybere pozici i v řetězu α zapsaném na druhé stopě. Přesněji, počínaje nejlevějším políčkem pásky, buď posune hlavu doprava anebo zvolí momentální pozici.
2. Nedeterministicky zvolí pravidlo $\beta \to \gamma$ gramatiky $\mathcal{G}$.
3. Je-li β podřetězem řetězu α, začínajícím na pozici i, pak nahradí β řetězem γ. V případě, že $|\gamma| < |\beta|$, „přisune“ zbývající část řetězu α tak, aby nově vytvořený řetěz byl zapsán na za sebou následujících políčkách. V situaci $|\gamma| > |\beta|$ je naopak zapotřebí zbývající část řetězu α„odsunout“, aby vznikl prostor pro zapsání celého řetězu γ.
4. Porovná vstupní řetěz, zapsaný na prvé stopě, s nově vytvořeným řetězem na druhé stopě. V případě rovnosti akceptuje, v případě neshody pokračuje bodem 1.

Lehce se prokáže, že každý řetěz vytvořený na druhé stopě je větnou formou gramatiky $\mathcal{G}$ a naopak, že každou větnou formu gramatiky $\mathcal{G}$ je možno popsaným způsobem vytvořit. Proto $L(\mathcal{G}) = L(\mathcal{M}) = L$. □

Lemma 4.19. *Nechť L je rekursivně spočetný jazyk. Pak L je generován gramatikou typu 0.*

Důkaz. Předpokládejme, že L je akceptován deterministickým Turingovym strojem $\mathcal{M} = (Q, \Sigma, \Gamma, \triangleright, \sqcup, \delta, q_0, q_{accept}, q_{reject})$. Naším cílem je zkonstruovat gramatiku $\mathcal{G}$ takovou, že slovo w se dá odvodit v $\mathcal{G}$, právě když w je akceptováno strojem $\mathcal{M}$. Proto odvození v gramatice musí nějakým způsobem simulovat výpočet stroje. Simulace je založena na tom, že v gramatice $\mathcal{G}$ vygenerujeme dvě kopie slova w nad abecedou Σ a nad druhou z nich v každém kroku odvození simulujeme jeden krok výpočtu stroje $\mathcal{M}$. Pokud stroj $\mathcal{M}$ neakceptuje, odvození nikdy nepovede k terminálnímu řetězu.

Formálně je $\mathcal{G} = (N, \Sigma, P, S)$, kde $N = \{S, S', K\} \cup Q \cup ((\Sigma \cup \{\varepsilon\}) \times \Gamma)$. Pro lepší názornost rozdělíme množinu pravidel do 3 skupin podle toho, ve které fázi odvozování je možno tato pravidla aplikovat.

I Na začátku odvození potřebujeme vygenerovat nějaké slovo w a jeho kopii (nad kterou se pak bude simulovat výpočet stroje $\mathcal{M}$). Proto budeme jako neterminály používat uspořádané dvojice $[x, y]$, podobně jako tomu bylo u TM s více stopami. Posloupnost takových neterminálů můžeme pak chápat jako dva řetězy zapsané nad sebou. Abychom mohli se spodním řetězem pracovat jako s konfigurací TM, přidáme neterminály odpovídající počátečnímu stavu stroje a levé koncové značce. Dále, abychom měli možnost rozpoznat, který symbol je poslední, přidáme na konec neterminál K.

1. $S \to q_0[\varepsilon, \triangleright]S'$
2. $S' \to [a, a]S'$ pro každé $a \in \Sigma$
3. $S' \to K$.

Aplikací pravidel 1-3 dokážeme v $\mathcal{G}$ vygenerovat z počátečního neterminálu S řetěz (pro lepší pochopení jsou uspořádané dvojice zapsány svisle)

$$q_0 \begin{bmatrix} \varepsilon \\ \triangleright \end{bmatrix} \begin{bmatrix} a_1 \\ a_1 \end{bmatrix} \begin{bmatrix} a_2 \\ a_2 \end{bmatrix} \cdots \begin{bmatrix} a_n \\ a_n \end{bmatrix} K$$

II Větná forma odvozena aplikací pravidel 1-3 obsahuje dvě informace. Na horní stopě je zapsáno slovo $w \in \Sigma^*$. Obsah dolní stopy spolu se symbolem q_0 určují počáteční konfiguraci výpočtu $\mathcal{M}$ na w (hlava snímá políčko bezprostředně vpravo od neterminálu určujícího stav). Druhá skupina pravidel umožní simulovat výpočet $\mathcal{M}$ na w. Přesněji, jestliže $\alpha \vdash_{\mathcal{M}} \beta$ je krok výpočtu stroje $\mathcal{M}$ a $\overline{\alpha}$ ($\overline{\beta}$) je větná forma obsahující na spodní stopě konfiguraci α (β), pak pravidla z druhé skupiny umožní odvození $\overline{\alpha} \Rightarrow_{\mathcal{G}} \overline{\beta}$.

4. $p[x, a] \to [x, b]\, q$
   pro každé $x \in \Sigma \cup \{\varepsilon\}$; $a, b \in \Gamma$; $p, q \in Q$ takové, že $\delta(p, a) = (q, b, R)$
5. $[y, c]p[x, a] \to q[y, c][x, b]$
   pro každé $x, y \in \Sigma \cup \{\varepsilon\}$; $a, b, c \in \Gamma$; $p, q \in Q$ takové, že $\delta(p, a) = (q, b, L)$
6. $pK \to [\varepsilon, b]qK$
   pro každé $b \in \Gamma$; $p, q \in Q$ takové, že $\delta(p, \sqcup) = (q, b, R)$
7. $[y, c]pK \to q[y, c][\varepsilon, b]K$
   pro každé $y \in \Sigma \cup \{\varepsilon\}$; $b, c \in \Gamma$; $p, q \in Q$ takové, že $\delta(p, \sqcup) = (q, b, L)$

Jestliže $\mathcal{M}$ akceptuje vstup $a_1 \cdots a_n$, pak

$$(q_0, \triangleright w \sqcup^\omega, 0) \vdash^*_{\mathcal{M}} (q_{accept}, \triangleright Z_1 \ldots Z_s \sqcup^\omega, i)$$

Použitím pravidel 4-7 můžeme v gramatice $\mathcal{G}$ odvodit

$$q_0 \begin{bmatrix} \varepsilon \\ \triangleright \end{bmatrix} \begin{bmatrix} a_1 \\ a_1 \end{bmatrix} \begin{bmatrix} a_2 \\ a_2 \end{bmatrix} \cdots \begin{bmatrix} a_n \\ a_n \end{bmatrix} K \Rightarrow^*_{\mathcal{G}} \begin{bmatrix} \varepsilon \\ \triangleright \end{bmatrix} \begin{bmatrix} a_1 \\ Z_1 \end{bmatrix} \cdots \begin{bmatrix} a_{i-1} \\ Z_{i-1} \end{bmatrix} q_{accept} \begin{bmatrix} a_i \\ Z_i \end{bmatrix} \cdots \begin{bmatrix} a_s \\ Z_s \end{bmatrix} K$$

kde $a_{n+1} = \ldots = a_s = \varepsilon$.

III Aplikací pravidel z prvních dvou skupin se dá v $\mathcal{G}$ odvodit řetěz ζ obsahující neterminál q_{accept} právě když $\mathcal{M}$ akceptuje slovo w zapsané na první stopě řetězu ζ. Třetí skupina pravidel umožní odvodit z ζ terminální řetěz. Přesněji, neterminály určující stav a konec (K) přepíšeme na ε. Neterminál — uspořádanou dvojici — přepíšeme na symbol z první komponenty dvojice.

8. $Zq_{accept} \to q_{accept}Z$ pro všechna $Z \in N$
9. $q_{accept}[a, x] \to aq_{accept}$ pro všechna $a \in \Sigma$, $x \in \Gamma$
10. $q_{accept}[\varepsilon, x] \to q_{accept}$ pro všechna $x \in \Gamma$
11. $q_{accept}K \to \varepsilon$

Množina pravidel gramatiky $\mathcal{G}$ je tvořena právě pravidly 1-11. □

Věta 4.20. *Třídy jazyků, které lze generovat gramatikami typu 0, resp. akceptovat Turingovymi stroji, jsou si rovny a tvoří právě třídu rekursivně spočetních jazyků.*

Pozorný čtenář si jistě v důkazu věty 4.19 povšimne, že libovolný rekursivně spočetný jazyk je tedy generovatelný gramatikou, která obsahuje právě jedno ε-pravidlo – viz pravidlo 11 (nejedná se však o pravidlo typu $S \to \varepsilon$, kde S se nevyskytuje na žádné pravé straně v pravidlech gramatiky – srv. definicí pojmu $\mathcal{G}$ je bez ε-pravidel).

4.6 Lineárně ohraničené automaty a jazyky typu 1

Výsledky prezentované v předchozí části ukazují, že rekursivně spočetné jazyky je možné popsat dvěma formalismy: prostřednictvím Turingových strojů a gramatik typu 0. Z výsledků předcházejících kapitol víme, že podobnou možnost máme i pro regulární jazyky (konečné automaty a regulární gramatiky) a bezkontextové jazyky (zásobníkové automaty a bezkontextové gramatiky). Zbývá najít protějšek kontextových gramatik. Ukážeme, že jím jsou Turingovy stroje se speciálním omezením na velikost pásky — tzv. lineárně ohraničené automaty.

Lineárně ohraničený automat (anglicky Linear Bounded Automaton), zkráceně LBA, je jednopáskový *nedeterministický* TM, který nikdy neopustí ta políčka, na kterých byl umístěn vstup. Formálně, lineárně ohraničený automat $\mathcal{M}$ je 10-tice

$$\mathcal{M} = (Q, \Sigma, \Gamma, \rhd, \lhd, \sqcup, \delta, q_0, q_{accept}, q_{reject})$$

kde symboly $Q, \Sigma, \Gamma, \rhd, \sqcup, \delta, q_0, q_{accept}, q_{reject}$ mají stejný význam jako u nedeterministického TM. $\lhd \in \Gamma \setminus \Sigma$ je *pravá koncová značka*. Podobně jako u TM požadujeme, aby LBA nikdy nepřepsal levou (pravou) koncovou značku jiným symbolem a aby nikdy neposunul svou hlavu vlevo (vpravo) od políčka obsahujícího levou (pravou) koncovou značku. Definice konfigurace, relace kroku výpočtu a jazyka $L(\mathcal{M})$ zůstávají stejné jako u nedeterministického TM.

Název LBA je odvozen z následujícího faktu. Uvažme ty nedeterministické TM $\mathcal{M}$, pro které existuje taková konstanta $k \in \mathbb{N}$, že pro každý vstupní řetěz w je pozice hlavy v každé konfiguraci každého výpočtu $\mathcal{M}$ na w nanejvýš $k \cdot |w|$. Zřejmě každý LBA splňuje uvedenou vlastnost. Naopak, ke každému TM uvedené vlastnosti se dá (technikou podobnou té, kterou jsme použili při simulaci k-páskového Turingova stroje jednopáskovým) zkonstruovat ekvivalentní LBA. Alternativně lze tedy definovat LBA jako TM s lineárně ohraničeným prostorem.

O LBA říkáme, že je *deterministický*, právě když pro každé $q \in Q$ a $a \in \Gamma$ je množina $\delta(q, a)$ jednoprvková. Není známé, zda třída jazyků akceptovaných deterministickými LBA je vlastní podtřídou třídy jazyků akceptovaných lineárně ohraničenými automaty. Víme, že každý jazyk akceptovaný nedeterministickým lineárně ohraničeným automatem se dá akceptovat deterministickým Turingovým strojem. Velikost pásky, kterou potřebuje takovýto Turingův stroj však může být až exponenciální funkcí délky vstupního slova a ne jenom lineární (viz konstrukce v 4.3).

Náš zájem o LBA je dán skutečností, že akceptují právě třídu kontextových jazyků. Důkaz tohoto tvrzení je podobný jako důkaz tvrzení, že gramatiky typu 0 akceptují právě třídu rekursivně spočetných jazyků.

Lemma 4.21. *Nechť L je jazyk generovaný kontextovou gramatikou. Pak L je akceptován nějakým lineárně ohraničeným automatem $\mathcal{M}$.*

Důkaz. Automat $\mathcal{M}$ konstruujeme podobně jako v důkazu lemmatu 4.18 s tím rozdílem, že nepovolíme, aby délka řetězu α na druhé stopě někdy přesáhla délku vstupního řetězu. Korektnost konstrukce plyne z faktu, že v kontextové gramatice pro odvození

$$S = w_0 \Rightarrow_{\mathcal{G}} w_1 \Rightarrow_{\mathcal{G}} w_2 \cdots \Rightarrow_{\mathcal{G}} w_n$$

platí $|w_{i-1}| \leq |w_i|$ pro každé $i = 1, \ldots, n$. Stroj $\mathcal{M}$ má proto dostatek prostoru na to, aby odvození vstupního řetězu, pokud existuje, našel. $\square$

Lemma 4.22. *Nechť $\mathcal{M}$ je lineárně ohraničený automat. Pak existuje kontextová gramatika generující jazyk $L(\mathcal{M})$.*

Důkaz. Důkaz tohoto tvrzení je nepatrnou modifikací důkazu lemmatu 4.19. Modifikace je nutná proto, že pravidla 10 a 11 nejsou kontextová. Změny doznají pravidla skupiny I, kdy první a poslední neterminál generovaného řetězu v sobě ponesou informaci o tom, že jsou prvním resp. posledním neterminálem. Neterminál, určující stav odpovídající konfigurace LBA, se stane součástí neterminálu pro obsah políčka. $\square$

Věta 4.23. *Třídy jazyků, které lze generovat kontextovými gramatikami, resp. rozhodovat lineárně ohraničenými automaty, jsou si rovny.*

Důležitou vlastností lineárně ohraničených automatů je, že každý LBA můžeme transformovat na ekvivalentní úplný Turingův stroj.

Věta 4.24. *Každý kontextový jazyk je rekursivní.*

Důkaz. Nechť L je jazyk akceptovaný lineárně ohraničeným automatem $\mathcal{M} = (Q, \Sigma, \Gamma, \rhd, \lhd, \sqcup, \delta, q_0, q_{accept}, q_{reject})$. Zkonstruujeme úplný LBA $\overline{\mathcal{M}}$ takový, že $L(\overline{\mathcal{M}}) = L$. Tím dokážeme tvrzení věty.

Konstrukce je založena na pozorování, že počet různých konfigurací, které se mohou vyskytnout ve výpočtu na vstupu w je konečný a jejich počet závisí jenom na délce slova w a na definici stroje $\mathcal{M}$. Když tedy stroj $\mathcal{M}$ slovo w akceptuje, pak nutně existuje akceptující výpočet $\mathcal{M}$ na w, jehož délka nepřesáhne počet různých konfigurací (v opačném případě se v něm nutně musí objevit dvě stejné konfigurace a vynecháním úseku mezi nimi skonstruujeme kratší akceptující výpočet). Proto stroji $\overline{\mathcal{M}}$ postačuje simulovat výpočet stroje $\mathcal{M}$ jenom do délky rovné počtu různých konfigurací.

Konfigurace stroje $\mathcal{M}$ v sobě nese informaci o stavu stroje, o obsahu pásky a o pozici hlavy. Počet různých konfigurací, které se mohou vyskytnout ve výpočtu na vstupu w délky n, je proto

$$|Q| \cdot |\Gamma|^n \cdot (n+2) \; .$$

Dále platí, že funkci $|Q| \cdot |\Gamma|^n \cdot (n+2)$ můžeme zhora ostře ohraničit funkcí c^n pro vhodně zvolené přirozené číslo c. Počet symbolů, které potřebujeme na zápis libovolného čísla z intervalu $0, \ldots, c^n - 1$ v c-ární číselné soustavě, nikdy nepřesáhne n.

LBA $\overline{\mathcal{M}}$ bude na vstupu w pracovat takto. Svoji pásku rozdělí na dvě stopy. Na horní stopě zůstane zapsán vstup w, na druhou zapíše číslo 0 (v c-ární číselné soustavě). Pak na horní stopě simuluje výpočet $\mathcal{M}$ na w, přičemž za každý odsimulovaný krok připočte k číslu, zapsanému na spodní stopě, jedničku (stále v c-ární číselné soustavě). Pokud $\mathcal{M}$ akceptuje (zamítne) vstup w, pak i $\overline{\mathcal{M}}$ akceptuje (zamítne). Když dojde k „přeplnění" spodní stopy, tak $\overline{\mathcal{M}}$ zamítne.

Výpočet LBA $\overline{\mathcal{M}}$ na libovolném vstupu w se zastaví buď proto, že simulovaný výpočet dosáhl akceptující resp. zamítající konfiguraci, nebo proto, že délka simulovaného výpočtu přesáhla hranici $c^{|w|}$. Jestliže tedy $\mathcal{M}$ akceptuje, pak existuje výpočet $\mathcal{M}$ na w takový, že jeho simulace přivede $\overline{\mathcal{M}}$ k akceptování. Naopak, jestliže $w \notin \mathcal{M}$, pak žádný výpočet $\overline{\mathcal{M}}$ na w nemůže být akceptující.

LBA $\overline{\mathcal{M}}$ je úplný, a proto jazyk L je rekursivní. □

V následující kapitole prezentujeme důkazovou techniku (tzv. metodu diagonalizace), která dovoluje dokázat, že ne každý rekursivní jazyk je nutně kontextový. V podstatě všechny jazyky, které považujeme za „přirozeně definované", jsou kontextové.

Pro úplnost ještě uveďme, jaké jsou uzávěrové vlastnosti třídy kontextových jazyků.

Věta 4.25. *Třída kontextových jazyků je uzavřena vzhledem k operacím* $\cup, \cap, \cdot, {}^*$ *a komplementu.*

Důkaz. Platnost tvrzení pro operace průniku, sjednocení, zřetězení a iterace se dokáže úplně stejným způsobem jako pro rekursivní jazyky (věta 4.15).

Důkaz pro komplement není tak přímočarý. Techniku použitou v důkazu věty 4.15 nemůžeme na LBA aplikovat. Problém je v tom, že LBA je definován jako *nedeterministické* výpočetní zařízení, a tedy pouhou záměnou akceptujícího a zamítajícího stavu bychom nezískali automat pro komplement jazyka (viz podobnou argumentaci pro nedeterministické konečné resp. zásobníkové automaty). Ve skutečnosti je důkaz uzavřenosti třídy kontextových jazyků na komplement dosti komplikovaný, a proto jej zde neuvádíme. □

Kapitola 5

Nerozhodnutelnost

Cílem této kapitoly je poskytnout odpověď na následující otázky:

1. Existuje jazyk (problém), který není rekursivně spočetný (částečně rozhodnutelný)?
2. Existuje jazyk (problém), který je rekursivně spočetný (částečně rozhodnutelný), ale není rekursivní (rozhodnutelný)?

Ukážeme, že odpověď na obě otázky je kladná. Z toho pak plyne další otázka:

3. Které problémy, týkající se jazyků Chomského hierarchie, jsou resp. nejsou rozhodnutelné?

Než začneme s úvahami o (ne)rozhodnutelnosti, projdeme dva okruhy problémů. Churchova teze ozřejmuje význam našich úvah o nerozhodnutelnosti. Na jejím základě můžeme totiž tvrzení o nerozhodnutelnosti konkrétního problému interpretovat jako tvrzení o *neexistenci algoritmu* řešícího uvažovaný problém. Podkapitola pojednávající o kódování TM je spíše technická a využijeme ji v následujících konstrukcích. Univerzální Turingův stroj je pro nás zajímavý hned ze dvou hledisek. Jednak jako technický nástroj, který můžeme využít tehdy, když potřebujeme, aby TM simuloval nějaký jiný TM. Z druhé strany, univerzální TM zdůrazňuje tu skutečnost, že Turingův stroj může být, podobně jako reálný počítač, programován.

5.1 Churchova teze

Cílem A. Turinga v době, kdy definoval svůj model (později nazvaný na jeho počest Turingův stroj), bylo rozlišit, co je a co není efektivní procedura, respektive jak bychom to řekli dnes, co je a co není algoritmus. Intuitivně, algoritmus je množina pravidel, které jednoznačně předepisují, co máme dělat pro to, abychom po konečném počtu kroků obdrželi požadovaný výsledek. Problém exaktní a jednoznačné definice pojmu algoritmus se stává klíčovým v okamžiku, když chceme dokázat, že neexistuje žádný algoritmus pro řešení konkrétního problému. Otázku existence resp. neexistence algoritmu si matematici kladli dávno (vzpomeňme alespoň problém kvadratury kruhu). Zvláště aktuální se stala v souvislosti s formulací známého Hilbertova programu začátkem našeho století. To vedlo k několika definicím pojmu algoritmus, z nichž jedna vychází právě z TM. Je známá pod názvem *Churchova teze*[1] (či též jako Church-Turingova teze):

> *Každý proces, který lze intuitivně nazvat algoritmem, se dá realizovat na úplném Turingově stroji.*

Obsahem Churchovy teze je tedy ztotožnění pojmů „algoritmicky řešitelný" a „řešitelný Turingovym strojem". Protože pojem algoritmicky řešitelný je jenom intuitivním pojmem, nemůže být obsah Churchovy téze formálně dokázán. Existuje však celá řada argumentů podporujících platnost teze, z nich uveďme například tyto:

1. Kromě TM bylo navrženo i mnoho jiných formalizmů; zmiňme alespoň
   - Postovy systémy
   - Minského stroje
   - μ-rekursivní funkce
   - λ-kalkul
   - while programy.

   Důležité je, že všechny se ukázaly, co se jejich výpočetní síly týče, jako vzájemně ekvivalentní.
2. Třída jazyků akceptovaných TM a třída funkcí počítaných TM jsou velice robustní. Různá omezení resp. rozšíření základního modelu nemají žádný vliv na tyto třídy (viz předcházející kapitola).
3. Doposud není znám žádný algoritmus, který by se nedal realizovat na Turingově stroji.

Více podrobností může čtenář najít literatuře věnované teorii vyčíslitelnosti.

Z pohledu Churovy teze je tedy problém[2] algoritmicky řešitelný, právě když je rozhodnutelný. V dalším textu budeme i nadále pracovat s Turingovými stroji, avšak budeme na ně nazírat především jako na algoritmy.

5.2 Kódování TM a univerzální TM

Základní vlastnost, ze které budeme v následujících úvahách vycházet, je schopnost Turingových strojů simulovat jiné Turingovy stroje, jejichž popis obdrží jako část svého vstupu.

První problém, na který při simulaci narazíme, je otázka vhodného kódování (zápisu) Turingových strojů. Požadujeme: 1. aby pro každý stroj (a případně i slovo nad nímž má pracovat), byl definován jeho kód (jako slovo nad nějakou abecedou) a 2. aby z kódu stroje (a případně i slova nad nímž má pracovat) byla jednoznačně dekódovatelná informace o jeho stavech, symbolech, přechodové funkci, ... (a případně i slovo nad nímž má pracovat).

Kódování Turingových strojů

Nechť $\mathcal{M} = (Q, \Sigma, \Gamma, \rhd, \sqcup, \delta, q_0, q_{accept}, q_{reject})$. Bez újmy na obecnosti můžeme předpokládat, že množina stavů $Q = \{q_0, q_1, q_2, \ldots, q_n\}$, přičemž $q_1 = q_{accept}$ je akcep-

1. Alonzo Church byl americký matematik
2. Jak je zřejmé již z kontextu, pod pojmem problém zde máme na mysli úlohu, kde řešením je buď ANO, nebo NE. Jakékoliv jiné úlohy, tj. takové, kde možných řešení je víc než jenom 2, můžeme nahlížet jako funkce a aplikovat na ně pojmy rekursivní resp. rekursivně spočetný.

tující a $q_2 = q_{reject}$ zamítající stav. Podobně $\Gamma = \{X_0, \ldots, X_z\}$, přičemž $X_0 = \triangleright$ je levá koncová značka a $X_1 = \sqcup$ je symbol pro prázdné políčko. Symbolům pro směr pohybu můžeme též přiřadit synonyma $L = S_1$ a $R = S_2$. Pak hodnotu přechodové funkce $\delta(q_i, X_j) = (q_k, X_l, S_m)$ můžeme jednoznačně kódovat binárním řetězem

$$0^i 1 0^j 1 0^k 1 0^l 1 0^m \ . \tag{5.1}$$

Binárním kódem Turingova stroje $\mathcal{M}$ je řetěz

$$111 \text{ kód}_1 \ 11 \text{ kód}_2 \ 11 \cdots 11 \text{ kód}_r \ 111 \ ,$$

kde kód$_i$ je řetězec tvaru 5.1 a kód$_1, \ldots,$ kód$_r$ jednoznačně popisují celou přechodovou funkci stroje $\mathcal{M}$[3]. Kód stroje $\mathcal{M}$ budeme označovat $\langle \mathcal{M} \rangle$. Každý binární řetězec je kódem nanejvýš jednoho Turingova stroje. Některé řetězy nejsou kódem žádného stroje; v takovém případě interpretujeme řetězec jako kód stroje přijímajícího prázdný jazyk.

Podobným způsobem můžeme kódovat i vstupní slova stroje $\mathcal{M}$ — slovu $X_{i_1} \cdots X_{i_k}$ přiřadíme kód $0^{i_1} 1 \cdots 1 0^{i_k} 1$. Prázdnému slovu přiřadíme kód ε. Kód slova w označujeme $\langle w \rangle$.

Příklad 5.1. *Mějme* $\mathcal{M} = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \{\triangleright, \sqcup, 0, 1, A, B\}, \delta, \triangleright, \sqcup, q_0, q_1, q_2)$ *s přechodovou funkcí*

$$\begin{aligned} \delta(q_0, \triangleright) &= (q_3, \triangleright, R), \\ \delta(q_3, 0) &= (q_3, A, R), \\ \delta(q_3, 1) &= (q_3, B, L), \\ \delta(q_3, A) &= (q_1, A, L), \\ \delta(q_3, \triangleright) &= (q_2, \triangleright, R). \end{aligned}$$

Kódem stroje $\mathcal{M}$ *je*

$$\begin{aligned} \langle \mathcal{M} \rangle = \; & 1111100011001100010010001000010011000100010001000001011 \\ & 0001000010100001011000110011001111 \ . \end{aligned}$$

Kódem slova 001101 je $\langle 001101 \rangle = 00100100010001001000.$

Univerzální Turingův stroj

Zavedené kódování strojů a jejich vstupů nám umožňuje zkonstruovat *univerzální Turingův stroj* $\mathcal{U}$ takový, že $\mathcal{U}$ akceptuje $\langle \mathcal{M} \rangle \sharp \langle w \rangle \overset{def}{\iff} \mathcal{M}$ akceptuje w, to jest:

$$L(\mathcal{U}) = \{ \ \langle \mathcal{M} \rangle \sharp \langle w \rangle \mid \mathcal{M} \text{ akceptuje } w \ \}.$$

Jinými slovy, jestliže stroj $\mathcal{U}$ dostane na vstup kód stroje $\mathcal{M}$ a kód jeho vstupu w (vzájemně odděleny symbolem $\sharp$), pak $\mathcal{U}$ akceptuje tento vstup, právě když $\mathcal{M}$ akceptuje w.
Stroj $\mathcal{U}$ pracuje takto:

---

3. Poznamenejme, že neklademe žádné podmínky na uspořádání, a proto jeden TM může mít více různých kódů.

1. nejdříve ověří, zda jeho vstup je tvaru $\{0,1\}^*\{\sharp\}\{0,1\}^*$. Když není, tak zastaví a zamítne.
2. $\mathcal{U}$ simuluje krok po kroku výpočet stroje $\mathcal{M}$ na w. Páska stroje $\mathcal{U}$ je rozdělena na tři stopy. Na první stopě si stroj $\mathcal{U}$ udržuje kód simulovaného stroje $\mathcal{M}$. Na druhé stopě má zaznamenán aktuální obsah pásky stroje $\mathcal{M}$ s vyznačenou pozicí hlavy. Třetí stopa slouží k zapamatování aktuálního stavu simulovaného stroje $\mathcal{M}$. Simulace jednoho kroku znamená, že stroj srovnává obsah třetí a relevantní části druhé stopy s obsahem první stopy, aby zjistil, jaká akce stroje $\mathcal{M}$ je předepsána pro jeho momentální stav a čtený symbol. Předepsanou akci odsimuluje takto: změní kód stavu na třetí stopě, přepíše snímaný symbol a posune značku pro pozici hlavy.
3. $\mathcal{U}$ akceptuje (zamítá), právě když se na třetí stopě objeví kód akceptujícího (zamítajícího) stavu. Pokud $\mathcal{M}$ na vstupu w cyklí, pak i $\mathcal{U}$ na vstupu $\langle\mathcal{M}\rangle\sharp\langle w\rangle$ cyklí.

Poznámka 5.2. *Na tomto místě (jako i na mnoha jiných v dalším textu) stavíme na Churchově tezi. Místo toho, abychom přesně definovali univerzální Turingův stroj (tj. specifikovali jeho množinu stavů, přechodovou funkci atd.), popsali jsme jeho činnost jen „slovně“. Důvod je zřejmý: tento popis je mnohem přehlednější a srozumitelnější, než kdybychom měli k dispozici popis přechodové funkce o délce několika (desítek) stran.*

5.3 Diagonalizace

Hlavním cílem této kapitoly je prozkoumat, které problémy, vztahující se k formálním jazykům a automatům, jsou resp. nejsou algoritmicky řešitelné. Popíšeme dvě matematické metody, které umožňují o problému dokázat, že není (částečně) rozhodnutelný — metodu diagonalizace a metodu redukce.

Jaký typ problémů je nerozhodnutelný, tj. algoritmicky neřešitelný? Jsou tyto problémy jen „teoretické“, nebo se s nimi můžeme běžně setkat? První z problémů, který prozkoumáme je formulován takto: máme daný program a přesnou specifikaci, co by tento program měl dělat (např. násobit dvě matice). Potřebujeme verifikovat, zda program skutečně dělá to, co od něj očekáváme. Jelikož jak program, tak i specifikace, jsou matematicky přesně definované objekty, přáli bychom si, aby proces verifikace byl pokud možno automatizován. Ukážeme, že právě toto je typ problému, který (obecně) není rozhodnutelný, a tedy řešitelný počítačem.

Přesněji formulováno, budeme se zabývat problémem příslušnosti pro Turinovy stroje. Pro daný Turingův stroj $\mathcal{M}$ a slovo w chceme určit, zda $\mathcal{M}$ slovo w akceptuje, nebo neakceptuje (tj. $\mathcal{M}$ zamítá w nebo na w cyklí). Jinými slovy, chceme určit, zda slovo w buď přísluší, nebo nepřísluší jazyku $L(\mathcal{M})$. Univerzální Turingův stroj a technika diagonalizace jsou nástroje, které nám umožní dokázat, že problém příslušnosti pro Turingovy stroje je nerozhodnutelný. Jinak řečeno, dokážeme, že jazyk

$$PP \stackrel{def}{=} \{\, \langle\mathcal{M}\rangle\sharp\langle w\rangle \mid \text{stroj } \mathcal{M} \text{ akceptuje } w\}$$

není rekursivní.

První idea, jak problém řešit, je použít univerzální stroj $\mathcal{U}$, kterému na vstup dáme řetěz $\langle\mathcal{M}\rangle\sharp\langle w\rangle$. $\mathcal{U}$ simuluje výpočet $\mathcal{M}$ na w a následně

- zastaví a akceptuje právě když $\mathcal{M}$ se zastaví a akceptuje
- zastaví a zamítne právě když $\mathcal{M}$ se zastaví a zamítne
- cyklí právě když $\mathcal{M}$ cyklí.

Lehce ověříme, že $L(\mathcal{U}) = PP$. Problém příslušnosti je tedy částečně rozhodnutelný. Protože však stroj $\mathcal{M}$ není úplný, pak ani $\mathcal{U}$ není úplný, a tedy z existence $\mathcal{U}$ neplyne rozhodnutelnost problému příslušnosti.

Na místě je tedy otázka, zda existuje nějaká jiná metoda, která na základě popisu Turingova stroje a jeho vstupu umožní vždy rozhodnout, jak dopadne výpočet stroje na daném vstupu. Ve skutečnosti takováto metoda neexistuje. Pro konkrétní TM a slovo se nám může podařit problém rozhodnout aplikací nějakého heuristického (ad hoc) přístupu, ale obecná metoda, která by poskytla řešení pro *libovolnou* dvojici ⟨ stroj, slovo ⟩ neexistuje.

Věta 5.3. *Problém příslušnosti pro Turingovy stroje je částečně rozhodnutelný, ale není rozhodnutelný.*

Částečnou rozhodnutelnost jsme již dokázali. Tvrzení o nerozhodnutelnosti dokážeme pomocí Cantorovy diagonalizační metody. Tuto metodu poprvé použil matematik Georg Cantor v roce 1873, když se zabýval problémem měření mohutnosti nekonečných množin. Diagonalizací prokázal, že množiny přirozených a reálných čísel mají různou mohutnost. Analogickým způsobem můžeme prokázat, že existuje jazyk, který není akceptován žádným Turingovým strojem.

Lemma 5.4. *Existuje jazyk, který není rekursivně spočetný.*

Důkaz. Z předcházejícího víme, že každý řetězec nad abecedou $\{0,1\}$ můžeme chápat jako kód nějakého Turingova stroje resp. jako kód nějakého slova. Pro každé $x \in \{0,1\}^*$ označme $\mathcal{M}_x$ Turingův stroj s kódem x. Podobně w_x je slovo, jehož kód je právě x. Rostoucí uspořádání slov[4] nad abecedou $\{0,1\}$ určuje uspořádání Turingových strojů

$$\mathcal{M}_\varepsilon, \mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_{00}, \mathcal{M}_{01}, \mathcal{M}_{10}, \mathcal{M}_{11}, \mathcal{M}_{000}, \ldots$$

a uspořádání slov

$$w_\varepsilon, w_0, w_1, w_{00}, w_{01}, w_{10}, w_{11}, w_{000}, \ldots \quad .$$

Snadno nahlédneme, že uvedené uspořádání je natolik jednoduché, že lze zkonstruovat úplný TM, který pro dané přirozené číslo m vypočte kód m-tého Turingova stroje resp. m-tého slova.

Uvažme nyní nekonečnou dvojrozměrnou tabulku (obr. 5.1). Řádky tabulky jsou označeny Turingovými stroji v zavedeném uspořádání. Sloupce jsou označeny slovy v zavedeném uspořádání. V průsečíku i-tého řádku a j-tého sloupce je v tabulce symbol 1, právě když i-tý Turingův stroj akceptuje j-té slovo. Jestliže stroj vstup neakceptuje, pak na uvažované pozici tabulka obsahuje symbol 0.

4. V rostoucím uspořádání slovo menší délky předchází slovu větší délky. Pokud $a = a_1 \cdot a_m$ a $b = b_1 \cdot b_m$ mají stejnou délku, pak a předchází b právě když existuje i takové, že $a_1 = b_1, \ldots, a_{i-1} = b_{i-1}$ a $a_i < b_i$.

	w_ε	w_0	w_1	w_{00}	w_{01}	w_{10}	w_{11}	w_{000}	w_{001}	$\cdots$
$\mathcal{M}_\varepsilon$	**1**	0	1	1	0	1	0	1	1	
$\mathcal{M}_0$	0	**1**	1	1	0	1	0	0	1	
$\mathcal{M}_1$	1	1	**1**	0	1	1	0	1	0	
$\mathcal{M}_{00}$	0	0	0	**0**	0	1	0	1	1	
$\mathcal{M}_{01}$	1	0	0	1	**0**	0	1	1	1	$\cdots$
$\mathcal{M}_{10}$	0	0	0	1	1	**1**	0	1	1	
$\mathcal{M}_{11}$	1	1	0	1	0	1	**0**	1	0	
$\mathcal{M}_{000}$	0	1	1	0	0	1	1	**1**	1	
$\mathcal{M}_{001}$	1	1	1	1	1	0	0	1	**0**	
$\vdots$					$\vdots$					$\ddots$

Obrázek 5.1: Tabulka obsahující informace o výpočtech Turingových strojů (symboly 1 a 0 jsou rozmístěny náhodně, čistě pro ilustraci.)

Zkonstruujeme jazyk D (tzv. diagonální jazyk) nad abecedou $\{0,1\}^*$ využitím prvků, které v tabulce leží na diagonále. Abychom zabezpečili, že jazyk D není akceptován žádným Turingovým strojem, požadujeme, aby slovo d patřilo do jazyka D tehdy a jen tehdy, když stroj $\mathcal{M}_d$ neakceptuje slovo w_d, tj. na průsečíku řádku $\mathcal{M}_d$ a sloupce w_d obsahuje tabulka symbol 0.

Lehce přijdeme ke sporu: předpokládejme, že existuje nějaký stroj $\mathcal{M}_x$ akceptující jazyk D. Jestliže slovo x patří do jazyka D, pak tabulka obsahuje na pozici $(\mathcal{M}_x, w_x)$ symbol 0 a nemůže být $L(\mathcal{M}_x) = D$. Naopak, jestliže slovo x nepatří do jazyka D, pak tabulka obsahuje na pozici $(\mathcal{M}_x, w_x)$ symbol 1, a tedy opět nemůže platit $L(\mathcal{M}_x) = D$. Proto neexistuje žádný TM, který by akceptoval jazyk D.

□

Nyní jsme již připraveni dokázat větu 5.3 o nerozhodnutelnosti problému příslušnosti.

Důkaz. **věty 5.3**
Důkaz věty provedeme sporem. Předpokládejme, že existuje *úplný* stroj $\mathcal{T}$ akceptující jazyk PP. Nad vstupním slovem $\langle M\rangle\sharp\langle w\rangle$ stroj $\mathcal{T}$ pracuje takto:

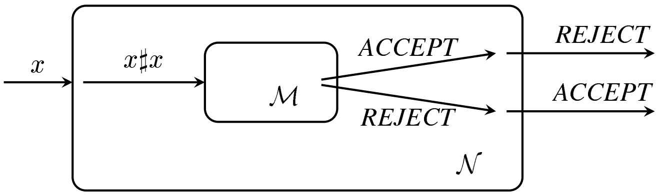
- $\mathcal{T}$ se zastaví a akceptuje právě když $\mathcal{M}$ se zastaví a akceptuje w
- $\mathcal{T}$ se zastaví a zamítne právě když $\mathcal{M}$ se zastaví a zamítne anebo když cyklí na w.

Zkonstruujeme nový Turingův stroj $\mathcal{N}$ (viz obr. 5.2), který pro vstup $x \in \{0,1\}^*$

1. zapíše na svou pásku řetěz $x\sharp x$,
2. simuluje výpočet stroje $\mathcal{T}$ na vstupu $x\sharp x$,
3. $\mathcal{N}$ akceptuje vstup x, právě když $\mathcal{T}$ zamítne vstup $x\sharp x$.
   $\mathcal{N}$ zamítne vstup x, právě když $\mathcal{T}$ akceptuje vstup $x\sharp x$.

Při konstrukci stroje $\mathcal{N}$ jsme využili existenci univerzálního TM. Konkrétně, v bodě 2. předepisujeme, aby stroj $\mathcal{N}$ simuloval jiný stroj, jehož popis dostal jako součást vstupu – to ale znamená, že se chová jako univerzální stroj. Pro každé slovo $x \in \{0,1\}^*$

$\mathcal{N}$ akceptuje $x \iff \mathcal{T}$ zamítá vstup $x\sharp x$ (podle definice $\mathcal{N}$)



Obrázek 5.2: Konstrukce stroje $\mathcal{N}$.

$$\iff \text{ stroj s kódem } x \text{ zamítá anebo cyklí na vstupu s kódem } x \quad \text{(podle předpokladu o } \mathcal{T}\text{)}$$

Speciálně nás zajímá výpočet stroje $\mathcal{N}$ na vstupu $\langle\mathcal{N}\rangle$, tj. na vstupu, který je kódem stroje $\mathcal{N}$. Tedy dostáváme

$$\begin{aligned}
\mathcal{N} \text{ akceptuje } \langle\mathcal{N}\rangle &\iff \mathcal{T} \text{ zamítá vstup } \mathcal{N}\sharp\mathcal{N} \\
&\iff \text{stroj s kódem } \langle\mathcal{N}\rangle \text{ zamítá anebo cyklí na vstupu s kódem } \langle\mathcal{N}\rangle \\
&\iff \mathcal{N} \text{ neakceptuje } \langle\mathcal{N}\rangle
\end{aligned}$$

To je zřejmý spor, a proto náš předpoklad o existenci úplného Turingova stroje $\mathcal{T}$ musel být chybný. □

Ještě jednou zopakujme, jakým způsobem jsme dokázali nerozhodnutelnost problému příslušnosti. Předpokládali jsme existenci stroje $\mathcal{T}$ rozhodujícího tento problém. Zkonstruovali jsme stroj $\mathcal{N}$ (který využíval $\mathcal{T}$) takový, že když $\mathcal{N}$ dostal na vstup x, tak ho akceptoval jedině tehdy, když stroj s kódem x neakceptoval vstup s kódem x. Nakonec jsme spustili stroj $\mathcal{N}$ na vstupu $\langle\mathcal{N}\rangle$. Chování stroje $\mathcal{N}$ popisuje řádek

	w_ε	w_0	w_1	w_{00}	w_{01}	w_{10}	w_{11}	w_{000}	w_{001}	$\cdots$
$\vdots$					$\vdots$					$\cdots$
$\mathcal{N}$	0	0	0	1	1	0	1	0	1	
$\vdots$					$\vdots$					$\ddots$

který vznikl „negací“ diagonály z tabulky 5.1. Srovnáme-li stroj $\mathcal{N}$ s libovolným Turingovym strojem $\mathcal{M}_x$, tak vidíme, že jejich chování na vstupu s kódem x se liší: když jeden ze strojů akceptuje, tak druhý neakceptuje a naopak.

Problém příslušnosti pro Turingovy stroje je příkladem problému, který je částečně rozhodnutelný, ale není rozhodnutelný. Přirozenou je otázka, zda existuje problém, který není ani částečně rozhodnutelný. Příkladem takového problému je komplement problému příslušnosti.

Věta 5.5. *Jazyk* co–$PP \stackrel{def}{=} \{\langle\mathcal{M}\rangle\sharp\langle w\rangle \mid \mathcal{M} \textit{ neakceptuje } w\}$ *není rekursivně spočetný.*

Důkaz. Předpokládejme, že jazyk co–PP je rekursivně spočetný. Pak podle věty 4.17 je jazyk PP rekursivní, což je spor. □

5.4 Redukce

S problémem příslušnosti pro TM úzce souvisí problém zastavení pro TM. Pro libovolný daný TM $\mathcal{M}$ a libovolné dané slovo w se ptáme, zda výpočet $\mathcal{M}$ na w je konečný či nikoli. Opět nás zajímá, zda je tento problém rozhodnutelný, tj. zda jazyk

$$PZ \stackrel{def}{=} \{\langle\mathcal{M}\rangle\sharp\langle w\rangle \mid \text{ výpočet } \mathcal{M} \text{ na } w \text{ je konečný } \}$$

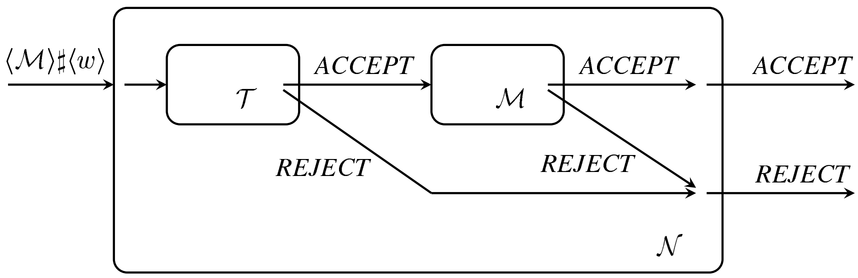
je rekursivní.

Věta 5.6. *Problém zastavení pro Turingovy stroje není rozhodnutelný.*

Důkaz. Důkaz provedeme sporem. Předpokládejme, že problém je rozhodnutelný. Pak existuje úplný Turingův stroj $\mathcal{T}$ akceptující jazyk PZ. Ukážeme, jak s pomocí stroje $\mathcal{T}$, můžeme sestrojit úplný Turingův stroj $\mathcal{N}$ rozhodující jazyk PP. Jelikož však jazyk PP není rekursivní, pak předpoklad o rozhodnutelnosti problému zastavení vede ke sporu.

Úkolem stroje $\mathcal{N}$ je pro daný stroj $\mathcal{M}$ a slovo w rozhodnout, zda $\mathcal{M}$ akceptuje w. Stroj $\mathcal{N}$ pro vstup $\langle\mathcal{M}\rangle\sharp\langle w\rangle$ pracuje takto (obr. 5.3):

1. simuluje výpočet stroje $\mathcal{T}$ na vstupu $\langle\mathcal{M}\rangle\sharp\langle w\rangle$,
2. v případě, že $\mathcal{T}$ akceptuje $\langle\mathcal{M}\rangle\sharp\langle w\rangle$, tak výpočet $\mathcal{M}$ na w je konečný. Proto $\mathcal{N}$ může simulovat výpočet $\mathcal{M}$ na w a $\mathcal{N}$ akceptuje právě když $\mathcal{M}$ akceptuje w,
3. v případě, že $\mathcal{T}$ zamítá $\langle\mathcal{M}\rangle\sharp\langle w\rangle$, tak $\mathcal{M}$ na w cyklí. Proto $\mathcal{N}$ zamítne svůj vstup.



Obrázek 5.3: Konstrukce stroje $\mathcal{N}$.

Stroj $\mathcal{N}$ je úplný, protože stroj $\mathcal{T}$ je úplný a $\mathcal{N}$ simuluje jen konečné výpočty stroje $\mathcal{M}$. Stroj $\mathcal{N}$ akceptuje jazyk PP, protože platí:

$\mathcal{N}$ akceptuje $\langle\mathcal{M}\rangle\sharp\langle w\rangle \iff$ stroj $\mathcal{T}$ akceptuje $\langle\mathcal{M}\rangle\sharp\langle w\rangle$ a $\mathcal{M}$ akceptuje w. □

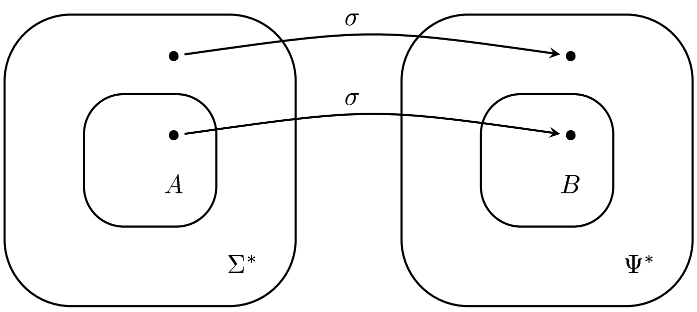
Metoda, kterou jsme použili v důkazu věty 5.6 je založena na tzv. *redukci*: problém příslušnosti jsme převedli (redukovali) na problém zastavení tak, že kdybychom měli k dispozici algoritmus řešící problém zastavení (stroj $\mathcal{T}$), pak bychom dokázali sestrojit i algoritmus rozhodující problém příslušnosti (stroj $\mathcal{N}$). Z předcházejícího víme, že takový algoritmus existovat nemůže, a tedy nemůže existovat ani algoritmus rozhodující problém zastavení. Jelikož úplně stejný postup můžeme použít i pro jiné problémy, zformulujeme metodu redukce a její použitelnost obecně.

Definice 5.7. Nechť A, B jsou jazyky, $A \subseteq \Sigma^*$, $B \subseteq \Psi^*$. *Redukce* jazyka A na jazyk B je rekursivní funkce $\sigma : \Sigma^* \to \Psi^*$ taková, že

$$w \in A \Longleftrightarrow \sigma(w) \in B \, .$$

V případě existence redukce jazyka A na jazyk B říkáme, že A je *redukovatelný* (se redukuje) na B a značíme $A \leq B$. S ohledem na známý vztah mezi pojmy jazyk a rozhodovací problém, můžeme aplikovat pojem redukce i na problémy. Zdůrazněme ještě jednou dvě klíčové vlastnosti redukce:

1. existence *úplného Turingova stroje* (algoritmu), který pro *každé* slovo w nad abecedou Σ vypočte jeho obraz, tj. slovo $\sigma(w)$ nad abecedou Ψ;
2. redukce zachovává příslušnost do jazyka (slovo z jazyka A se zobrazí na slovo z jazyka B, slovo nepatřící jazyku A se zobrazí na slovo nepatřící jazyku B) (obr. 5.4).



Obrázek 5.4: Redukce.

Způsob, jakým lze pojem redukce využít při důkazu o (ne)rozhodnutelnosti nějakého problému, je vyjádřen v následující větě.

Věta 5.8. *Nechť $A \leq B$.*
(i) Není-li jazyk A rekursivně spočetný, pak ani jazyk B není rekursivně spočetný.
(ii) Není-li jazyk A rekursivní, pak ani jazyk B není rekursivní.

Alternativní (a ekvivalentní) formulace věty 5.8 je
Nechť $A \leq B$.
$(\bar{i})$ Je-li jazyk B rekursivně spočetný, pak i jazyk A je rekursivně spočetný.
$(\overline{ii})$ Je-li jazyk B rekursivní, pak i jazyk A je rekursivní.

Důkaz. **(i)** Dokážeme alternativu $(\bar{i})$. Tvrzení (i) dostaneme kontrapozicí implikace. Předpokládejme, že $A \leq B$ a že B je rekursivně spočetný. Nechť $\mathcal{R}$ je úplný TM počítající redukci σ jazyka A na jazyk B. Dále nechť $\mathcal{T}_B$ je TM akceptující jazyk B. Sestrojíme nový TM $\mathcal{T}_A$ akceptující jazyk A a tím dokážeme, že A je rekursivně spočetný.

Stroj $\mathcal{T}_A$ pro vstup w (viz obr. 5.5) pracuje takto:

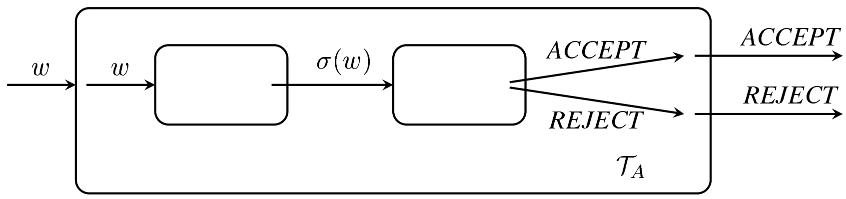
1. simuluje výpočet stroje $\mathcal{R}$ na vstupu w. Výsledkem simulace je řetěz $\sigma(w)$;
2. simuluje výpočet stroje $\mathcal{T}_B$ na vstupu $\sigma(w)$;
3. pokud stroj $\mathcal{T}_B$ zastaví a akceptuje (zamítne), pak i $\mathcal{T}_A$ zastaví a akceptuje (zamítne). Když $\mathcal{T}_B$ cyklí, pak i $\mathcal{T}_A$ cyklí.

Tedy platí:

$$\begin{aligned}\mathcal{T}_A \text{ akceptuje } w &\Longleftrightarrow \mathcal{T}_B \text{ akceptuje } \sigma(w)\\ &\Longleftrightarrow \sigma(w) \in B\\ &\Longleftrightarrow w \in A\end{aligned}$$

(ii) Analogicky jako v předcházejícím případě. Protože však B je rekursivní, existuje úplný TM $\mathcal{T}_B$, který ho akceptuje. Pak ale i stroj $\mathcal{T}_A$ bude úplný, a tedy jazyk A rekursivní.

$\mathcal{T}_B$ □



Obrázek 5.5: Konstrukce stroje $\mathcal{T}_A$.

Důkaz nerozhodnutelnosti problému P metodou redukce se skládá ze dvou kroků.

1. Zvolíme nějaký problém N, o kterém už bylo dokázáno, že je nerozhodnutelný.
2. Prokážeme, že $\mathsf{N} \leq \mathsf{P}$.

Nerozhodnutelnost problému P je pak důsledkem věty 5.8.

Redukci můžeme, samozřejmě, využít i k důkazu rozhodnutelnosti nějakého problému P. V takovém případě

1. Zvolíme nějaký problém R, o kterém už bylo dokázáno, že je rozhodnutelný.
2. Prokážeme, že $\mathsf{P} \leq \mathsf{R}$.

Rozhodnutelnost problému P je pak opět důsledkem věty 5.8.

Konstrukce redukce $A \leq B$ se skládá z následujících kroků. Nechť $A \subseteq \Sigma^*$, $B \subseteq \Psi^*$.

1. Definujeme funkci $\sigma : \Sigma^* \to \Psi^*$.
2. Ověříme, že funkce σ je rekursivní (například tak, že zkonstruujeme úplný Turingův stroj (tj. algoritmus), který pro každé $w \in \Sigma^*$ vypočte $\sigma(w)$).
3. Ověříme platnost ekvivalence

$$w \in A \Longleftrightarrow \sigma(w) \in B\ .$$

Otázka 5.9. *Který z jazyků PP, PZ hraje v důkazu věty 5.6 roli jazyka A a který roli jazyka B. Jak je definována redukce A na B?*

Otázka 5.10. *Ukažte, že redukovatelnost je tranzitivní, tj. když $A \leq B$ a $B \leq C$, pak $A \leq C$. Je redukovatelnost symetrická, tj. plyne z $A \leq B$ platnost $B \leq A$?*

5.5 Další rozhodnutelné a nerozhodnutelné problémy pro TM

Doposud jsme se setkali se třemi nerozhodnutelnými problémy, z nichž dva (problém zastavení a příslušnosti pro TM) byly částečně rozhodnutelné a třetí (komplement problému zastavení) nebyl ani částečně rozhodnutelný. Dále jsme ukázali princip, jak lze pomocí redukce dokázat (částečnou) rozhodnutelnost či nerozhodnutelnost jiných problémů. Aplikujme nyní tyto poznatky a prozkoumejme další problémy týkající se TM (rekursivně spočetných jazyků).

Věta 5.11 (Rozhodnutelné problémy)**.** *Následující problémy jsou rozhodnutelné.*
Pro libovolný daný Turingův stroj $\mathcal{M}$ rozhodnout, zda
(a) *$\mathcal{M}$ má alespoň 1998 stavů,*
(b) *výpočet stroje $\mathcal{M}$ nad vstupním slovem a^{1998} je delší než 1998,*
(c) *existuje slovo w takové, že výpočet stroje $\mathcal{M}$ nad vstupním slovem w je delší než 1998.*

Věta 5.12 (Semirozhodnutelné problémy)**.** *Následující problémy nejsou rozhodnutelné, ale jsou částečně rozhodnutelné.*
Pro libovolný daný Turingův stroj $\mathcal{M}$ rozhodnout, zda
(a) *jazyk $L(\mathcal{M})$ je neprázdný,*
(b) *jazyk $L(\mathcal{M})$ obsahuje alespoň 1998 slov.*

Věta 5.13 (Nerozhodnutelné problémy)**.** *Následující problémy nejsou (ani) částečně rozhodnutelné.*
Pro libovolný daný Turingův stroj $\mathcal{M}$ rozhodnout, zda
(a) *jazyk $L(\mathcal{M})$ je prázdný,*
(b) *jazyk $L(\mathcal{M})$ obsahuje nanejvýš 1998 slov,*
(c) *jazyk $L(\mathcal{M})$ je konečný,*
(d) *jazyk $L(\mathcal{M}) = R$ pro libovolný* daný *regulární jazyk R,*
(e) *jazyk $L(\mathcal{M})$ je regulární (tj. zda* existuje *regulární jazyk R takový, že $L(\mathcal{M}) = R$),*
(f) *jazyk $L(\mathcal{M})$ je rekursivní (tj. zda existuje rekursivní jazyk L takový, že $L(\mathcal{M}) = L$, tj. problém, zda $\mathcal{M}$ je úplný TM).*

Důkaz. věty 5.11
Rozhodnutelnost všech uvedených problémů prokážeme tak, že zkonstruujeme úplný TM $\mathcal{T}$ akceptující právě kódy Turingových strojů majících požadovanou vlastnost.

(a) Stroj $\mathcal{T}$ prochází vstup a testuje, zdali je na některé z pozic příslušejících stavům (viz kódování TM) řetěz tvaru $0^{1998}0^*$.

(b) Stroj $\mathcal{T}$ má tři pásky. Na třetí pásce si na počátku výpočtu označí 1999 políček. Na druhou pásku zapíše řetěz a^{1998}. Pak na druhé pásce simuluje krok po kroku výpočet stroje s kódem x (x je vstup stroje $\mathcal{T}$) na vstupu a^{1998}. Za každý odsimulovaný krok označí $\mathcal{T}$ jeden symbol na třetí pásce. Když simulovaný výpočet skončil dříve, než byly označeny všechny symboly na třetí pásce, tak $\mathcal{T}$ zamítá. Pokud $\mathcal{T}$ označil všechny symboly, tak akceptuje.

(c) Naším cílem je zkonstruovat TM $\mathcal{T}$, který pro dané $\langle\mathcal{M}\rangle$ rozhodne, zdali existuje slovo w takové, že výpočet stroje $\mathcal{M}$ na vstupu w je delší než 1998. Stroj $\mathcal{T}$ bere postupně slova nad vstupní abecedou stroje $\mathcal{M}$ v rostoucím uspořádání až do délky 1999. Pro každé slovo simuluje výpočet stroje $\mathcal{M}$ nad tímto slovem, přičemž si pamatuje počet už odsimulovaných kroků výpočtu. Když délka simulovaného výpočtu přesáhne 1998, tak $\mathcal{T}$ se zastaví a akceptuje. Když délka výpočtu na žádném z uvažovaných slov nepřesáhne 1998, tak $\mathcal{T}$ se zastaví a vstup zamítne.

Zůstává prokázat korektnost navrženého postupu. Tvrdíme, že když jsme hledané slovo nenašli mezi slovy délky maximálně 1999, tak skutečně neexistuje. Víme, že délka výpočtu na žádném ze slov délky 1999 nepřesáhla hodnotu 1998. To znamená, že stroj nikdy nečetl poslední symbol vstupu (na to by potřeboval alespoň 1999 kroků). Průběh výpočtu je tedy jednoznačně určen prefixem délky 1998 a *není* ovlivněn symboly za tímto prefixem.

□

Důkaz. věty 5.12

(a) Nejdříve dokážeme, že problém neprázdnosti není rozhodnutelný. Navrhneme redukci jazyka PZ (který není rozhodnutelný — věta 5.6) na jazyk

$$PN \stackrel{def}{=} \{ \langle \mathcal{M} \rangle \mid L(\mathcal{M}) \neq \emptyset \}.$$

Nerozhodnutelnost problému neprázdnosti plyne z věty 5.8.

Protože $PZ \subseteq \{0,1,\sharp\}^*$, $PN \subseteq \{0,1\}^*$, tak hledaná redukce σ je funkce z $\{0,1,\sharp\}^*$ do $\{0,1\}^*$. Slovu $x \in \{0,1,\sharp\}^*$ přiřadí σ slovo $\langle \mathcal{R}_x \rangle$, přičemž $\langle \mathcal{R}_x \rangle$ je kód takto definovaného Turingova stroje. Stroj $\mathcal{R}_x$ pro vstup $w \in \{0,1\}^*$ pracuje následovně.

1. Jestliže slovo x nepatří do $\{0,1\}^*\{\sharp\}\{0,1\}^*$, tak $\mathcal{R}_x$ vstup w zamítne.
2. V opačném případě smaže obsah své pásky a zapíše na ní slovo x. Nechť $x = x_1 \sharp x_2$.
3. Stroj $\mathcal{R}_x$ simuluje výpočet stroje s kódem x_1 na vstupu s kódem x_2.
4. Jestliže simulovaný výpočet je konečný, tak $\mathcal{R}_x$ akceptuje. Pokud simulovaný výpočet je nekonečný, tak i výpočet $\mathcal{R}_x$ je nekonečný.

Funkce σ je úplná a je vyčíslitelná (vypočítatelná) Turingovým strojem, což znamená, že je rekursivní. Je důležité si uvědomit, že jazyk akceptovaný strojem $\mathcal{R}_x$ je

$$L(\mathcal{R}_x) = \begin{cases} \emptyset & \text{když stroj s kódem } x_1 \text{ na vstupu s kódem } x_2 \text{ cyklí,} \\ & \text{resp. v případě že } x \text{ nepatří do } \{0,1\}^*\{\sharp\}\{0,1\}^* \\ \{0,1\}^* & \text{když výpočet stroje s kódem } x_1 \text{ na vstupu} \\ & \text{s kódem } x_2 \text{ je konečný.} \end{cases}$$

Funkce σ zachovává příslušnost do jazyka, protože

$$\begin{aligned} \langle \mathcal{R}_x \rangle \in PN &\iff L(\mathcal{R}_x) = \{0,1\}^* \\ &\iff x = x_1 \sharp x_2,\ x_1, x_2 \in \{0,1\}^* \text{ a} \\ &\ \text{výpočet stroje s kódem } x_1 \text{ na vstupu s kódem } x_2 \text{ je konečný} \\ &\iff x \in PZ \end{aligned}$$

Zůstává dokázat, že jazyk PN je rekursivně spočetný. Zkonstruujeme Turingův stroj $\mathcal{T}$ akceptující jazyk PN. Nabízí se vcelku přímočaré řešení. Chceme-li zjistit, zda daný stroj $\mathcal{M}$ akceptuje vůbec nějaké slovo, stačí brát všechna možná vstupní slova a simulovat postupně výpočet stroje $\mathcal{M}$ na každém z nich. Pokud $\mathcal{M}$ akceptuje nějaké

slovo, určitě na něj dříve nebo později narazíme. Potíž je v tom, že dříve než dojde na toto slovo, které by stroj $\mathcal{M}$ akceptoval, tak se může zkoušet i slovo, na němž $\mathcal{M}$ cyklí – k hledanému slovu se tak nikdy nedostaneme.
Potíž můžeme obejít využitím „paralelismu". Namísto toho, aby se simuloval vždy jen jeden výpočet stroje $\mathcal{M}$, bude stroj $\mathcal{T}$ simulovat několik výpočtů stroje $\mathcal{M}$ najednou. Provedeme to tak, že $\mathcal{T}$ vždy odsimuluje krok jednoho výpočtu, pak krok dalšího výpočtu atd. (Uvedená metoda nese anglický název dovetailing.)
Předpokládejme libovolné, ale fixní uspořádání slov nad vstupní abecedou stroje $\mathcal{M}$. Pracovní páska stroje $\mathcal{T}$ bude rozdělena na několik úseků oddělených speciálním symbolem $\$ \in \Gamma$. V každém úseku je aktuální konfigurace jednoho simulovaného výpočtu. Na počátku má $\mathcal{T}$ jen jeden úsek a na něm počáteční (nultou) konfiguraci na prvním vstupu stroje $\mathcal{M}$. Jeden cyklus spočívá v tom, že $\mathcal{T}$ prochází svoji pracovní pásku zleva doprava. V každém úseku přepíše konfiguraci jejím následovníkem. Když dojde za poslední úsek, napíše tam počáteční konfiguraci výpočtu na dalším, ještě neprozkoumaném slově. Obsah pracovní pásku stroje $\mathcal{T}$ po pátém opakování cyklu je schematicky naznačen na obrázku 5.6 (symbol $Konfig_j^i$ značí j-tou konfiguraci výpočtu stroje $\mathcal{M}$ na i-tém vstupu). Pokud se v některém z úseků objeví

$\triangleright$	$Konfig_4^1$	$\$$	$Konfig_3^2$	$\$$	$Konfig_2^3$	$\$$	$Konfig_1^4$	$\$$	$Konfig_0^5$	$\$$	$\sqcup \ldots$

Obrázek 5.6: Paralelní simulace výpočtů

akceptující konfigurace (což nastane, právě když jazyk $L(\mathcal{M})$ je neprázdný), pak stroj $\mathcal{T}$ akceptuje. Pro vstup $\langle\mathcal{M}\rangle$ takový, že $L(\mathcal{M}) = \emptyset$, stroj $\mathcal{T}$ cyklí.

(b) Tvrzení dokážeme nepatrnou modifikací předcházejícího důkazu. Pro nerozhodnutelnost stačí vzít v úvahu, že jazyk $L(\mathcal{M})$ obsahuje alespoň 1998 slov tehdy a jenom tehdy, když $L(\mathcal{R}_x) = \{0,1\}^*$. Pro semirozhodnutelnost stačí modifikovat stroj $\mathcal{T}$ tak, aby akceptoval až po objevení se 1998 akceptujících konfigurací.

$\square$

Důkaz. věty 5.13

(a) Nerozhodnutelnost problému prázdnosti (anglicky emptiness) dokážeme redukcí jazyka co–PZ (který není rekursivně spočetný — dokažte!) na jazyk PE, kde

$$PE \stackrel{def}{=} \{\, \langle\mathcal{M}\rangle \mid L(\mathcal{M}) = \emptyset\}.$$

Redukce σ je definována podobným způsobem, jako v důkazu nerozhodnutelnosti případu neprázdnosti (věta 5.12, případ (a)). Jediná změna se týká bodu 1.: pokud slovo x nepatří do $\{0,1\}^*\{\sharp\}\{0,1\}^*$, pak $\mathcal{R}_x$ vstup w *akceptuje*. Opět platí, že jazyk $L(\mathcal{R}_x)$ je buď prázdný (v případě, že stroj s kódem x_1 na vstupu s kódem x_2 cyklí) , nebo stroj $\mathcal{R}_x$ akceptuje každé vstupní slovo (to v případě, že výpočet stroje s kódem x_1 na vstupu s kódem x_2 je konečný, resp. v případě že x nepatří do $\{0,1\}^*\{\sharp\}\{0,1\}^*$). Funkce σ zachovává příslušnost do jazyka, protože

$$\begin{aligned}\langle \mathcal{R}_x \rangle \in PE &\iff L(\mathcal{R}_x) = \emptyset \\ &\iff x = x_1 \sharp x_2,\ x_1, x_2 \in \{0,1\}^* \text{ a} \\ &\qquad \text{stroj s kódem } x_1 \text{ na vstupu s kódem } x_2 \text{ cyklí} \\ &\iff x \in \text{co}-PZ\end{aligned}$$

(b),(c) Stejně jako předcházející případ. Jazyk $L(\mathcal{R}_x)$ je buď prázdný, a tedy obsahuje nanejvýš 1998 slov, resp. je konečný, nebo je nekonečný.

(d) Předpokládejme, že problém, zda daný TM a konečný automat akceptují stejný jazyk, je částečně rozhodnutelný. Pak za konečný automat můžeme zvolit automat akceptující prázdný jazyk a dostáváme, že i problém prázdnosti pro Turingovy stroje je částečně rozhodnutelný — spor.

(e) Zvolme jazyk L, který je bezkontextový a není regulárný. Nechť $\mathcal{A}$ je zásobníkový automat akceptující jazyk L. Chceme dokázat, že jazyk

$$PR \stackrel{def}{=} \{\, \langle \mathcal{M} \rangle \mid L(\mathcal{M}) \text{ je regulární} \,\}.$$

není rekursivně spočetný. Důkaz provedeme redukcí jazyka co$-PZ$ na jazyk PR. Hledaná redukce σ slovu $x \in \{0,1,\sharp\}^*$ přiřadí slovo $\langle \mathcal{R}_x \rangle$, přičemž $\langle \mathcal{R}_x \rangle$ je kód takto definovaného Turingova stroje. Stroj $\mathcal{R}_x$ pro vstup $w \in \{0,1\}^*$ pracuje následovně.

1. Jestliže slovo x nepatří do $\{0,1\}^*\{\sharp\}\{0,1\}^*$, tak $\mathcal{R}_x$ pokračuje bodem 4.
2. V opačném případě změní svou pásku na třístopou. Na druhou stopu zapíše slovo x. Nechť $x = x_1 \sharp x_2$.
3. Stroj $\mathcal{R}_x$ simuluje na své třetí stopě výpočet stroje s kódem x_1 na vstupu s kódem x_2. Jestliže simulovaný výpočet je konečný, tak $\mathcal{R}_x$ pokračuje bodem 4. Pokud simulovaný výpočet stroje s kódem x_1 na vstupu s kódem x_2 je nekonečný, tak i výpočet $\mathcal{R}_x$ je nekonečný.
4. $\mathcal{R}_x$ simuluje výpočet zásobníkového automatu $\mathcal{A}$ na vstupu w.
5. Jestliže automat $\mathcal{A}$ slovo w akceptuje, tak i stroj $\mathcal{R}_x$ svůj vstup w akceptuje. Pokud $\mathcal{A}$ zamítne, tak i $\mathcal{R}_x$ zamítá.

Funkce σ je úplná a je vyčíslitelná Turingovým strojem, což znamená, že je rekursivní. Jazyk akceptovaný strojem $\mathcal{R}_x$ je

$$L(\mathcal{R}_x) = \begin{cases} L & \text{jestliže výpočet stroje s kódem } x_1 \text{ na vstupu s kódem } x_2 \text{ je konečný} \\ & \quad \text{resp. v případě že } x \text{ nepatří do } \{0,1\}^*\{\sharp\}\{0,1\}^*) \\ \emptyset & \text{jestliže stroj s kódem } x_1 \text{ na vstupu s kódem } x_2 \text{ cyklí} \end{cases}$$

Funkce σ zachovává příslušnost do jazyka, protože

$$\begin{aligned}\langle \mathcal{R}_x \rangle \in PR &\iff L(\mathcal{R}_x) = \emptyset \\ &\iff x = x_1 \sharp x_2,\ x_1, x_2 \in \{0,1\}^* \text{ a} \\ &\qquad \text{stroj s kódem } x_1 \text{ na vstupu s kódem } x_2 \text{ cyklí} \\ &\iff x \in \text{co}-PZ\end{aligned}$$

(f) Analogicky jako v případě (e) s tím rozdílem, že jako jazyk L zvolíme jazyk, který je rekursivně spočetný a není rekursivní.

□

5.6 Postův korespondenční problém

V předchozí části jsme zkoumali rozhodnutelnost různých problémů týkajících se Turingových strojů. Poznali jsme, že až na několik málo velice jednoduchých problémů, jsou nerozhodnutelné. Přirozenou je proto otázka, co se stane, když v uvedených problémech zaměníme TM nějakým výpočtově slabším zařízením. Překvapujícím(?) je zjištění, že pokud chceme, aby se problém stal rozhodnutelným, pak ho musíme většinou (až na několik málo již ukázaných výjimek) formulovat pro velice omezenou třídu jazyků: pro deterministické bezkontextové, resp. v některých případech až pro regulární jazyky. Shrnutí všech uvažovaných problémů najde čtenář na konci této kapitoly.

V předchozí části byl klíčovým důkaz nerozhodnutelnosti problému příslušnosti (věta 5.3). Nerozhodnutelnost všech ostatních problémů byla dokázána redukcí. Obdobnou klíčovou roli v této části sehraje tzv. Postův korespondeční problém. Postův problém je úzce spjat s problémem zastavení, avšak jeho formulace je pro naše cíle mnohem vhodnější.

Formulace

Postův korespondenční problém (anglicky Post Correspondence Problem), zkráceně PKP (PCP), lze formulovat takto: jsou dány dva seznamy, $A = x_1, \ldots, x_n$ a $B = y_1, \ldots, y_n$, neprázdných slov nad abecedou Σ. Seznamy A, B nazýváme *instancí* (případem) PKP a označujeme $\langle A, B\rangle$. Daná instance PKP *má řešení*, právě když existuje konečná posloupnost přirozených čísel $i_1, i_2, \ldots i_k$, $k \geq 1$, taková, že

$$x_{i_1} x_{i_2} \cdots x_{i_k} = y_{i_1} y_{i_2} \cdots y_{i_k} .$$

Posloupnost $i_1, i_2, \ldots i_k$ se nazývá *řešením* PKP. Postův korespondenční problém je formulován jako třída problémů rozhodnout pro libovolnou danou instanci PKP, zda má řešení.

Příklad 5.14. *Nechť A, B jsou seznamy nad abecedou $\{a, b, c\}$,*

$$A = (b, cbb, ab, c) \qquad B = (bbc, b, a, bc).$$

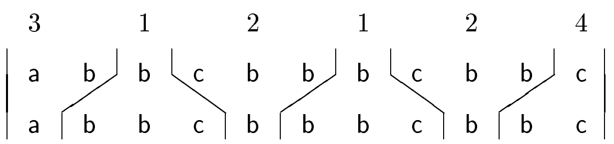
Pro lepší představu si instanci PKP můžeme znázornit jako kostky domina

$$\left\langle \frac{A}{B} \right\rangle = \left\{ \left[\frac{\mathsf{b}}{\mathsf{bbc}}\right], \left[\frac{\mathsf{cbb}}{\mathsf{b}}\right], \left[\frac{\mathsf{ab}}{\mathsf{a}}\right], \left[\frac{\mathsf{c}}{\mathsf{bc}}\right] \right\}$$

Řešením uvedené instance PKP je posloupnost 3,1,2,1,2,4 protože

$$x_3 x_1 x_2 x_1 x_2 x_4 = y_3 y_1 y_2 y_1 y_2 y_4 = abbcbbbcbbc.$$

Opět pro lepší představu uvádíme i grafickou prezentaci



Uvedená posloupnost není jediným řešením daného případu; jsou jím např. i posloupnosti 3,1,2,1,2,1,2,4 a 3,1,2,1,2,4,3,1,2,1,2,4 a další.

Otázka 5.15. *Kolik řešení má instance PKP z předcházejícího problému?*

Příklad 5.16. *Mějme instanci PKP, danou seznamy A, B nad abecedou $\{0, 1\}$ takto:*

$$A = (01, 001, 11) \qquad B = (10, 00, 011).$$

Hledané řešení by muselo začínat indexem 2, protože dvojice x_1, y_1 a též x_3, y_3 se liší v již prvním symbolu (přesněji: v žádné z těchto dvojic není jedno ze slov předponou druhého). Tím máme

$$\begin{array}{|ccc}
0 & 0 & 1 \\
0 & 0 &
\end{array}$$

Ze seznamu B musíme nyní vybrat slovo, které začíná symbolem 1. Jedinou možností je slovo 10. Dostáváme

$$\begin{array}{|ccccc}
0 & 0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 &
\end{array}$$

a to je situace shodná s předcházející. Proto tato instantce PKP nemá řešení (neexistuje konečná *posloupnost čísel požadovaných vlastností).*

Jsme tedy schopni pro některé konkrétní případy rozhodnout, zda mají řešení. Jak ovšem uvidíme, nelze napsat algoritmus, který by pro libovolnou instanci PKP rozhodoval, zda má či nemá řešení.

Iniciální Postův korespondeční problém

Našim cílem je dokázat, že Postův korespondenční problém není rozhodnutelný. Použijeme metodu redukce a sestrojíme redukci problému příslušnosti pro TM (věta 5.3) na PKP. Protože sestrojit hledanou redukci přímo je poměrně (technicky) náročné, zjednodušíme si celý problém následovně. Definujeme *iniciální* Postův korespondenční problém (zkráceně inPKP) a prokážeme, že pokud je inPKP nerozhodnutelný, tak i PKP je nerozhodnutelný. Pak dokážeme, že inPKP je nerozhodnutelný.

Rozdíl ve formulaci inPKP a PKP je v tom, že u iniciálního Postova problému se pro danou instanci $\langle A, B\rangle$ ptáme, zda má řešení začínající číslem 1. Přesněji, instance $\langle A, B\rangle$ iniciálního Postova korespondečního problému má řešení právě když existuje posloupnost přirozených čísel $i_1, i_2, \ldots i_k$, $k \geq 0$, taková, že

$$x_1 x_{i_1} x_{i_2} \cdots x_{i_k} = y_1 y_{i_1} y_{i_2} \cdots y_{i_k} .$$

Příklad 5.17. *Iniciální Postův korespondeční problém pro seznamy A, B z příkladu 5.14 má řešení 122, protože $x_1 x_1 x_2 x_2 = y_1 y_1 y_2 y_2 =$ bbcbbcbb.*

Lemma 5.18. *Z nerozhodnutelnosti iniciálního Postova korespondečního problému plyne nerozhodnutelnost Postova korespondečního problému.*

Důkaz. Předpokládejme, že PKP je rozhodnutelný. Dané instanci $\langle A, B\rangle$ inPKP přiřadíme instanci $\langle C, D\rangle$ PKP tak, že $\langle A, B\rangle$ má řešení právě když $\langle C, D\rangle$ má řešení. Z rozhodnutelnosti PKP by tedy plynula i rozhodnutelnost inPKP.

Nechť tedy seznamy $A = x_1, \dots, x_n$ a $B = y_1, \dots, y_n$ nad abecedou Σ jsou instancí iniciálního Postova problému. Dále nechť $\$, \phi$ jsou dva symboly nepatřící do abecedy Σ. Zavedeme homomorfismy $h_L, h_R : \Sigma^* \to \Sigma^* \cup \{\$, \phi\}$ definované předpisem: $h_L(a) \stackrel{def}{=} \phi a,\ h_R(a) \stackrel{def}{=} a\phi$ pro všechna $a \in \Sigma$ (s přirozeným rozšířením ze Σ na Σ^*). Položme

$$\begin{array}{lll} X_1 = \phi h_R(x_1) & Y_1 = h_L(y_1) & \\ X_{i+1} = h_R(x_i) & Y_{i+1} = h_L(y_i) & \text{pro } 1 \le i \le n \\ X_{n+2} = \$ & Y_{n+2} = \phi\$ & \end{array}$$

Seznamy C a D nyní vytvoříme takto:

$$C = (X_1, \dots, X_{n+2}) \qquad\qquad D = (Y_1, \dots, Y_{n+2}).$$

Ověřme, že instance $\langle C, D\rangle$ PKP má řešení, právě když instance $\langle A, B\rangle$ inPKP má řešení:
1. $\Longleftarrow$: Nechť řešením instance $\langle A, B\rangle$ inPKP je posloupnost $i_1, i_2, \dots i_k$. Protože

$$\phi h_R(x_1 x_{i_1} \cdots x_{i_k})\$ = h_L(y_1 y_{i_1} \cdots y_{i_k})\phi\$,$$

je posloupnost $1, (i_1 + 1), \dots (i_k + 1), n + 2$ řešením instance $\langle C, D\rangle$ PKP.
2. $\Longrightarrow$: Nechť řešením instance $\langle C, D\rangle$ PKP je posloupnost $j_1, j_2, \dots, j_k$. Pak nutně musí být $j_1 = 1$ a $j_k = n + 2$. Řešením instance $\langle A, B\rangle$ inPKP pak bude například posloupnost $(j_2 - 1), \dots, (j_l - 1)$, kde l je nejmenší takové číslo, že $j_{l+1} = n + 2$. Nutnost volby l je dána faktem, že dvojice X_{n+2}, Y_{n+2} nemá v $\langle A, B\rangle$ žádný vzor. $\square$

Příklad 5.19. *Redukcí instance $\langle A, B\rangle$ iniciálního PKP*

$$\left\langle \frac{A}{B} \right\rangle = \left\{ \left[\frac{\mathsf{ba}}{\mathsf{b}}\right], \left[\frac{\mathsf{b}}{\mathsf{bb}}\right], \left[\frac{\mathsf{b}}{\mathsf{abb}}\right], \left[\frac{\mathsf{bab}}{\mathsf{a}}\right] \right\} \quad \textit{dostaneme instanci } \langle C, D\rangle \textit{ PKP}$$

$$\left\langle \frac{C}{D} \right\rangle = \left\{ \left[\frac{\phi\mathsf{b}\phi\mathsf{a}\phi}{\phi\mathsf{b}}\right], \left[\frac{\mathsf{b}\phi\mathsf{a}\phi}{\phi\mathsf{b}}\right], \left[\frac{\mathsf{b}\phi}{\phi\mathsf{b}\phi\mathsf{b}}\right], \left[\frac{\mathsf{b}\phi}{\phi\mathsf{a}\phi\mathsf{b}\phi\mathsf{b}}\right], \left[\frac{\mathsf{b}\phi\mathsf{a}\phi\mathsf{b}\phi}{\phi\mathsf{a}}\right], \left[\frac{\$}{\phi\$}\right] \right\}$$

Řešení 2,4,2 instance $\langle A, B\rangle$ odpovídá například řešení 1,3,5,3,6 instance $\langle C, D\rangle$.

Poznámka 5.20. *Jiná formulace tvrzení uvedeného v lemmatu 5.18 je, že inPKP $\le$ PKP. Dokažte, že platí i opačné tvrzení, tj. že PKP $\le$ inPKP.*

Nerozhodnutelnost Postova korespondenčního problému

Věta 5.21. *Postův korespondenční problém je nerozhodnutelný.*

Důkaz. Vzhledem k tvrzení lemmatu 5.18 stačí dokázat nerozhodnutelnost iniciálního Postova problému. Sestrojíme redukci problému příslušnosti pro TM (věta 5.3) na iniciální Postův problém: dvojici Turingův stroj $\mathcal{T}$ a slovo w tato redukce přiřadí dva seznamy A a B tak, že instance $\langle A, B\rangle$ inPKP má řešení, právě když stroj $\mathcal{T}$ akceptuje slovo w.

Nechť $\mathcal{T} = (Q, \Sigma, \Gamma, \rhd, \sqcup, \delta, q_0, q_{accept}, q_{reject})$, $w \in \Sigma^*$. Dále předpokládejme, že $Q \cap \Gamma = \emptyset$ a že $\sharp \notin Q \cup \Gamma$ (nový symbol). Každou konfiguraci (q, z, r) stroje $\mathcal{T}$ můžeme jednoznačně reprezentovat řetězcem $z_1 q z_2$, kde $z_1 z_2 \sqcup^\omega = z$ a $|z_1| = r$. V této reprezentaci je pozice hlavy určena umístěním symbolu stavu v řetězci z. Základní idea konstrukce je přiřadit dvojici $\mathcal{T}$, w takovou instanci iniciálního Postova problému, že její

	Seznam A	Seznam B	
I	$\sharp$	$\sharp q_0 \rhd w \sharp$	
II	Z	Z	pro všechna $Z \in \Gamma$
	$\sharp$	$\sharp$	
III	pro všechna $q \in Q \setminus \{q_{accept}\},\ p \in Q,\ X, Y, Z \in \Gamma$		
	qX	Yp	jestliže $\delta(q, X) = (p, Y, R)$
	ZqX	pZY	jestliže $\delta(q, X) = (p, Y, L)$
	$q\sharp$	$Yp\sharp$	jestliže $\delta(q, \sqcup) = (p, Y, R)$
	$Zq\sharp$	$pZY\sharp$	jestliže $\delta(q, \sqcup) = (p, Y, L)$
IV	Zq_{accept}	q_{accept}	pro všechna $Z \in \Gamma$
	$q_{accept}Z$	q_{accept}	pro všechna $Z \in \Gamma$
V	$q_{accept}\sharp\sharp$	$\sharp$	

Obrázek 5.7: Redukce problému příslušnosti pro TM na inPKP

řešení (posloupnost čísel) určuje slovo $\sharp \rhd q_0 w \sharp \alpha_1 q_1 \beta_1 \sharp \cdots \sharp \alpha_k q_{accept} \beta_k \sharp\sharp$ takové, že jeho předpona je zápisem akceptujícího výpočtu $\mathcal{T}$ na w (za podmínky, že existuje).

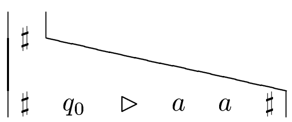
Seznamy A, B jsou tvořeny slovy nad abecedou $\Sigma \cup \Gamma \cup \{\sharp\}$ tak, jak je uvedeno v tabulce 5.7. Pro lepší pochopení jsou slova seskupena do menších celků. S výjimkou první dvojice (skupina I), která musí být v seznamech na prvním místě, uspořádání zbylých dvojic může být libovolné. Konstrukci nejdříve ilustrujeme na příkladu a až pak prokážeme její korektnost.

Příklad 5.22. *Prezentovanou redukci ilustrujeme na příkladu stroje* $\mathcal{T} = (\{q_0, q_1\}, \{a, b\}, \{\rhd, \sqcup, a, b, A\}, \rhd, \sqcup, \delta, q_0, q_{accept}, q_{reject})$ *(δ je dána tabulkou 5.1) a slova* $w = aa$*. Dvo-*

	$\rhd$	a	A	$\sqcup$
q_0	$(q_0, \rhd, R)$	(q_1, A, R)	(q_1, A, L)	(q_{accept}, A, R)
q_1	$(q_{reject}, \rhd, R)$	(q_0, A, R)	(q_0, A, L)	$(q_0, \sqcup, L)$

Tabulka 5.1: Přechodová funkce stroje $\mathcal{T}$

jici $\mathcal{T}$ a w přiřadíme případ $\langle A, B\rangle$ iniciálního PKP popsaný v tabulce 5.8. Pokusíme se najít řešení této instance PKP. Jako první musíme vzít ze seznamu A slovo $\sharp$ a ze seznamu B k němu odpovídající slovo $\sharp q_0 \rhd aa\sharp$ (plyne z definice iniciálního PKP).

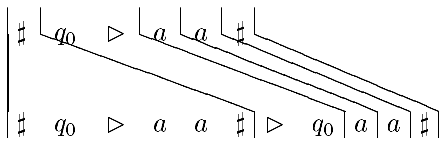


Ze seznamu A musíme dále vybírat tak, abychom vytvořili řetěz $q_0 \rhd aa\sharp$. K dispozici

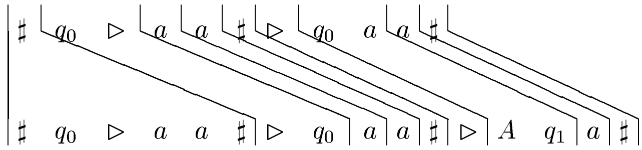
Seznam A	Seznam B	
$\sharp$	$\sharp q_0 aa \sharp$	
$\rhd$	$\rhd$	
$\sqcup$	$\sqcup$	
a	a	
b	b	
A	A	
$\sharp$	$\sharp$	
$q_0 \rhd$	$\rhd q_0$	protože $\delta(q_0, \rhd) = (q_0, \rhd, R)$
$q_0 a$	$A q_1$	protože $\delta(q_0, a) = (q_1, A, R)$
$\rhd q_0 A$	$q_1 \rhd A$	protože $\delta(q_0, A) = (q_1, A, L)$
$a q_0 A$	$q_1 a A$	
$A q_0 A$	$q_1 A A$	
$\sqcup q_0 A$	$q_1 \sqcup A$	
$q_0 \sqcup$	$A q_{accept}$	protože $\delta(q_0, \sqcup) = (q_{accept}, A, R)$
$q_0 \sharp$	$A q_{accept} \sharp$	
$q_1 \rhd$	$\rhd q_{reject}$	protože $\delta(q_1, \rhd) = (q_{reject}, \rhd, R)$
$q_1 a$	$A q_0$	protože $\delta(q_1, a) = (q_0, A, R)$
$\rhd q_1 A$	$q_0 \rhd A$	protože $\delta(q_1, A) = (q_0, A, L)$
$a q_1 A$	$q_0 a A$	
$A q_1 A$	$q_1 A A$	
$\sqcup q_1 A$	$q_1 \sqcup A$	
$\rhd q_1 \sqcup$	$q_0 \rhd \sqcup$	protože $\delta(q_1, \sqcup) = (q_0, \sqcup, L)$
$a q_1 \sqcup$	$q_0 a \sqcup$	
$A q_1 \sqcup$	$q_0 A \sqcup$	
$\sqcup q_1 \sqcup$	$q_0 \sqcup \sqcup$	
$\rhd q_1 \sharp$	$q_0 \rhd \sharp$	
$a q_1 \sharp$	$q_0 a \sharp$	
$A q_1 \sharp$	$q_0 A \sharp$	
$\sqcup q_1 \sharp$	$q_0 \sqcup \sharp$	
$\rhd q_{accept}$	q_{accept}	
$a q_{accept}$	q_{accept}	
$A q_{accept}$	q_{accept}	
$\sqcup q_{accept}$	q_{accept}	
$q_{accept} \rhd$	q_{accept}	
$q_{accept} a$	q_{accept}	
$q_{accept} A$	q_{accept}	
$q_{accept} \sqcup$	q_{accept}	
$q_{accept} \sharp\sharp$	$\sharp$	

Obrázek 5.8: Seznamy A a B

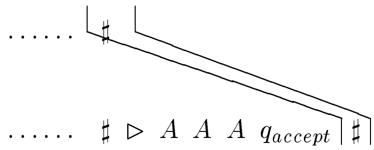
máme slova z druhé a třetí skupiny.



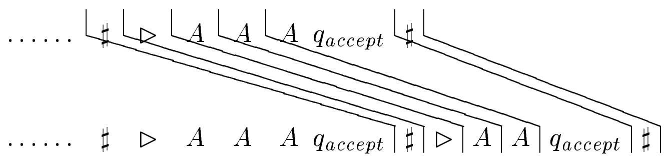
Všimněme si, že zatímco první slovo jsme prodloužili o řetěz $q_0 \triangleright aa\sharp$ (počáteční konfigurace $\mathcal{T}$ na w), k druhému slovu jsme přidali řetěz $\triangleright q_0 aa\sharp$, což je konfigurace, do které přejde $\mathcal{T}$ z počáteční konfigurace v jednom kroku výpočtu. Opět tedy musíme k prvnímu slovu přidat $\triangleright q_0 aa\sharp$.



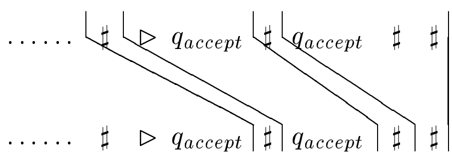
Podobně postupujeme, dokud nenastane situace



Symbol q_{accept} ve spodním slově ukazuje, že stroj $\mathcal{T}$ by slovo aa akceptoval. Proto bychom měli být schopni najít řešení dané instance iniciálního PKP. Skutečně, dvojice ze 4. a 5. skupiny nám umožní, aby se obě slova „srovnala“:



Podobně postupujeme až do úspěšného konce



Pokračování důkazu věty 5.21 Máme dokázat, že navržená transformace je redukcí. Rekursivita je zřejmá. Zůstává ověřit, že Turingův stroj $\mathcal{T}$ akceptuje slovo w tehdy a jen tehdy, když k němu přiřazená instance iniciálního PKP má řešení.

Je-li $\langle(x_1, \ldots, x_n), (y_1, \ldots, y_n)\rangle$ instance iniciálního PKP, pak posloupnost indexů $i_1, \ldots, i_m$ nazveme *částečným řešením* této instance, právě když slovo $x = x_1 x_{i_1} \cdots x_{i_m}$ je *prefixem* slova $y = y_1 y_{i_1} \cdots y_{i_m}$. O slovech x a y říkáme, že jsou *definovány* částečným řešením $i_1, \ldots, i_m$.

Nechť $q_0 \rhd w \vdash u_1q_1v_1 \vdash u_2q_2v_2 \vdash \cdots \vdash u_kq_kv_k$ je výpočet stroje $\mathcal{T}$ na vstupu w a nechť $q_k \notin \{q_{accept}, q_{reject}\}$. Tvrdíme, že pak existuje částečné řešení instance $\langle A, B\rangle$ definující dvojici slov (x, y),

$$
\begin{aligned}
x &= \sharp q_0 \rhd w \sharp u_1q_1v_1\sharp \cdots \sharp u_{k-1}q_{k-1}v_{k-1}\sharp \\
y &= \sharp q_0 \rhd w \sharp u_1q_1v_1\sharp \cdots \sharp u_{k-1}q_{k-1}v_{k-1}\sharp u_kq_kv_k\sharp
\end{aligned}
$$

a navíc, že neexistuje žádné jiné řešení definující dvojici slov (c, y) pro žádné c.

Uvedené tvrzení lehce dokážeme indukcí vzhledem ke k. Pro $k = 0$ je tvrzení triviální: jedině prázdná posloupnost čísel definuje dvojici slov $(\sharp, \sharp q_0 w)$.

Předpokládejme, že tvrzení platí pro nějaké k a že $q_k \notin \{q_{accept}, q_{reject}\}$. Dokážeme, že pak platí i pro $k+1$. Protože $y = xz$, kde $z = u_kq_kv_k\sharp$, tak ze seznamu A musíme vybrat slova tvořící z. Pro symboly $Z \in \Gamma$ můžeme použít jedině slova ze skupiny II. Pro symbol q_k a symbol bezprostředně za ním následující resp. předcházející je ve skupině III jediné slovo. Toto slovo spolu se svým protějškem ze seznamu B přirozeným způsobem prezentují krok výpočtu stroje $\mathcal{T}$. Žádný jiný výběr neumožňuje vytvoření slova z.

Tímto dostáváme nové častečné řešení definující dvojici slov $(y, yu_{k+1}q_{k+1}v_{k+1}\sharp)$. Lehce nahlédneme, že $u_kq_kv_k \vdash u_{k+1}q_{k+1}v_{k+1}$. Navíc, jestliže $q_{k+1} = q_{accept}$, tak jednoduchým způsobem získáme (použitím slov ze skupin IV a V) z tohoto částečného řešení řešení instance $\langle A, B\rangle$.

Tedy existuje-li akceptující výpočet stroje $\mathcal{T}$ na vstupu w, pak instance $\langle A, B\rangle$ iniciálního PKP má řešení. Pokud $\mathcal{T}$ slovo w neakceptuje, pak $\langle A, B\rangle$ může mít částečná řešení, která ale definují dvojice slov nestejné délky a není možné je prodloužit na řešení. $\square$

5.7 Nerozhodnutelné problémy z teorie formálních jazyků

Nerozhodnutelnost Postova korespondenčního problému využijeme v následujících úvahách o (ne)rozhodnutelnosti některých problémů týkajících se gramatik Chomského hierarchie. Poznamenáváme jenom, že vzhledem ke známým vztahům mezi gramatikami a automaty, všechna uvedená tvrzení platí stejně i pro odpovídající typ automatů.

Problém prázdnosti

Je formulován jako úloha pro libovolnou danou gramatiku $\mathcal{G}$ rozhodnout, zda $L(\mathcal{G}) = \emptyset$.

Věta 5.23. *Problém prázdnosti pro třídu* bezkontextových *gramatik je* rozhodnutelný.

Důkaz. Je obsažen v důkazu věty 3.9. $\square$

Důsledek 5.24. *Problém prázdnosti pro třídu* regulárních *gramatik je* rozhodnutelný.

Důkaz. Protože každá regulární gramatika je současně bezkontextovou, na rozhodování problému můžeme použít algoritmus navržený pro bezkontextové gramatiky. $\square$

Úvahu použitou v důkazu důsledku 5.24 můžeme lehce zobecnit. Nechť P je problém a S množina jeho instancí. Pak z rozhodnutelnosti problému P pro množinu instnací S plyne rozhodnutelnost tohoto problému pro každou množinu instancí S', kde $\mathsf{S}' \subseteq \mathsf{S}$. Tvrzení *nemusí* platit pro množinu instancí S'', kde $\mathsf{S}'' \supset \mathsf{S}$, což dokazuje následující věta.

Věta 5.25. *Problém prázdnosti pro třídu* kontextových *gramatik je* nerozhodnutelný.

Důkaz. Navrhneme redukci komplementu Postova korespondenčního problému na problém prázdnosti pro kontextové gramatiky. Komplement PKP není rozhodnutelný (kdyby byl, tak podle věty 4.16 i PKP by byl rozhodnutelný). Redukce přiřadí seznamům A, B kontextovou gramatiku $\mathcal{G}$ takovou, že instance $\langle A, B\rangle$ nemá řešení tehdy a jen tehdy, když gramatika $\mathcal{G}$ generuje prázdný jazyk.

Vyjdeme z poznatku, že daná instance $A = (x_1, \ldots, x_n), B = (y_1, \ldots, y_n)$ PKP nad abecedou Σ buď nemá žádné řešení, nebo jich má nekonečně mnoho. Uvažme jazyky L_A a L_B nad abecedou $\Sigma \cup \{\sharp, 1, \ldots, n\}$ (předpokládáme $\Sigma \cap \{\sharp, 1, \ldots, n\} = \emptyset$);

$$L_A \stackrel{def}{=} \{\, x_{i_1} \cdots x_{i_k} \sharp i_k \cdots i_1 \mid 1 \le i_j \le n \text{ pro } j = 1, \ldots, k\}$$
$$L_B \stackrel{def}{=} \{\, y_{i_1} \cdots y_{i_k} \sharp i_k \cdots i_1 \mid 1 \le i_j \le n \text{ pro } j = 1, \ldots, k\}.$$

Průnik těchto dvou jazyků $L_A \cap L_B$ obsahuje právě ta slova $u\sharp v$, pro která $v = i_k \cdots i_1$ a posloupnost $i_1, \ldots, i_k$ je řešením instance $\langle A, B\rangle$ Postova problému. Lehce ověříme, že jazyky L_A a L_B jsou kontextové (jsou dokonce determinininistické bezkontextové), a tedy i jejich průnik $L_A \cap L_B$ je kontextový jazyk (věta 4.25). Nechť $\mathcal{G}$ je kontextová gramatika generující jazyk $L_A \cap L_B$. Hledaná redukce přiřadí instanci $\langle A, B\rangle$ gramatiku $\mathcal{G}$. □

Důsledek 5.26. *Problém prázdnosti pro třídu* gramatik typu 0 *je* nerozhodnutelný.

Důkaz. Z rozhodnutelnosti problému pro třídu gramatik typu 0 by plynula i rozhodnutelnost problému pro třídu kontextových gramatik, což je spor. □

Použitou úvahu můžeme opět formulovat obecně: z nerozhodnutelnosti problému P pro množinu instancí S plyne nerozhodnutelnost tohoto problému pro každou množinu instancí S', kde $\mathsf{S}' \supseteq \mathsf{S}$.

Problém příslušnosti

Je formulován jako úloha rozhodnout pro libovolnou danou gramatiku $\mathcal{G}$ a libovolné dané slovo w, zda $w \in L(\mathcal{G})$. S tímto problémem jsme se setkali již jednou (věta 5.3) a víme tedy, že pro gramatiky typu 0 je nerozhodnutelný. Z lemmatu 4.21 plyne, že ke každé kontextové gramatice je možné zkonstruovat ekvivalentní *úplný* Turingův stroj, a proto problém příslušnosti pro kontextové gramatiky je rozhodnutelný.

Problém konečnosti

Je formulován jako úloha pro libovolnou danou gramatiku $\mathcal{G}$ rozhodnout, zda jazyk $L(\mathcal{G})$ je konečný.

Věta 5.27. *Problém konečnosti pro bezkontextové gramatiky je rozhodnutelný.*

Důkaz. Tvrzení je důsledkem lemmatu o vkládáni pro CFL (věta 3.24). Jazyk $L(\mathcal{G})$ je nekonečný tehdy a jen tehdy, když obsahuje slovo z délky $p < |z| \le p + q$. □

Věta 5.28. *Problém konečnosti pro kontextové gramatiky je nerozhodnutelný.*

Důkaz. Důkaz tvrzení je obsažen v důkazu věty 5.25. Stačí si uvědomit, že jazyk $L_A \cap L_B$ obsahuje právě slova odpovídající řešením PKP, a tedy je buď prázdný (PKP nemá řešení), nebo nekonečný (PKP má řešení, tedy jich má nekonečně mnoho). □

Další nerozhodnutelné problémy

Nerozhodnutelnost budeme opět dokazovat redukcí z PKP resp. komplementu PKP. Využijeme označení zavedené v důkazu věty 5.25, tj. seznamy $A = (x_1, \ldots, x_n)$, $B = (y_1, \ldots, y_n)$ nad abecedou Σ tvoří instanci PKP, $\Sigma \cap \{\sharp, 1, \ldots, n\} = \emptyset$. K nim jsou přiřazeny jazyky

$$L_A \stackrel{def}{=} \{\, x_{i_1} \cdots x_{i_k} \sharp i_k \cdots i_1 \;\mid\; 1 \le i_j \le n \text{ pro } j = 1, \ldots, k\}$$
$$L_B \stackrel{def}{=} \{\, y_{i_1} \cdots y_{i_k} \sharp i_k \cdots i_1 \;\mid\; 1 \le i_j \le n \text{ pro } j = 1, \ldots, k\}.$$

jejichž průnik je neprázdný, právě když instance $\langle A, B\rangle$ má řešení. Navíc definujeme jazyky $L_{A,B}$ a S předpisem

$$L_{A,B} \stackrel{def}{=} L_A \cdot \{\sharp\} \cdot L_B^{\mathsf{R}}$$
$$S \stackrel{def}{=} \{\, u\sharp v\sharp v^{\mathsf{R}}\sharp u^{\mathsf{R}} \;\mid\; u \in \Sigma^*, v \in \{1, \ldots, n\}^* \,\}.$$

Otázka 5.29. *Sestrojte deterministické zásobníkové automaty akceptující jazyky $L_{A,B}$ a S.*

Vlastnosti definovaných jazyků popisují následující tři lemmata.

Lemma 5.30. *Instance $\langle A, B\rangle$ Postova problému má řešení, právě když*

$$L_{A,B} \cap S \neq \emptyset$$

Důkaz. Předpokládejme, že posloupnost čísel $i_1, \ldots, i_k$ je řešením instance $\langle A, B\rangle$. Tato posloupnost určuje slova $u \in S$ a $v \in L_{A,B}$, přičemž

$$u = x_{i_1} \cdots x_{i_k} \sharp i_k \cdots i_1 \sharp i_1 \cdots i_k \sharp x_{i_k} \cdots x_{i_1}$$
$$v = x_{i_1} \cdots x_{i_k} \sharp i_k \cdots i_1 \sharp i_1 \cdots i_k \sharp y_{i_k} \cdots y_{i_1}.$$

Protože $i_1, \ldots, i_k$ je řešením instance $\langle A, B\rangle$, je $x_{i_1} \cdots x_{i_k} = y_{i_1} \cdots y_{i_k}$ a následně $u = v$. Jazyk $L_{A,B} \cap S$ je tedy neprázdný. Opačná implikace se dokáže analogicky. □

Lemma 5.31. *Jazyk* $\mathsf{co}\text{–}(L_{A,B} \cap S)$ *je bezkontextový.*

Důkaz. Vyjděme ze vztahu $\mathsf{co}\text{–}(L_{A,B} \cap S) = \mathsf{co}\text{–}L_{A,B} \cup \mathsf{co}\text{–}S$. Protože jazyk $L_{A,B}$ je deterministický bezkontextový (5.29), je i jazyk $\mathsf{co}\text{–}(L_{A,B})$ deterministický bezkontextový (věta 3.82). Analogicky pro jazyk S. Tvrzení lemmatu plyne z uzavřenosti třídy bezkontextových jazyků vůči operaci sjednocení (věta 3.58). □

Lemma 5.32. *Jazyk $L_{A,B} \cap S$ je bezkontextový tehdy a jen tehdy, když je prázdný.*

Důkaz. Pro danou instanci $\langle A, B\rangle$ Postova problému obsahuje (podle lemmatu 5.30) jazyk $L_{A,B} \cap S$ právě slova odpovídající řešením instance $\langle A, B\rangle$. Protože každá instance PKP buď nemá žádné řešení nebo jich má nekonečně mnoho, tak i jazyk $L_{A,B} \cap S$ je buď prázdný anebo nekonečný. Pokud $L_{A,B} \cap S$ je prázdný, tak je triviálně bezkontextový (ba dokonce regulární).

Abychom ukázali obrácenou implikaci, předpokládejme, že $L_{A,B} \cap S$ je neprázdný a že je bezkontextový. Pro každý bezkontextový jazyk platí lemma o vkládání (věta 3.24). Tedy i pro $L_{A,B} \cap S$ musí existovat dvě konstanty p, q takové, že každé slovo $z \in (L_{A,B} \cap S)$ délky větší než p lze napumpovávat (existence takovéhoto slova plyne z nekonečnosti jazyka $L_{A,B} \cap S$). Zvolme slovo $z \in L$, $z = x_{i_1} \cdots x_{i_k} \sharp i_k \cdots i_1 \sharp i_1 \cdots i_k \sharp y_{i_k} \cdots y_{i_1}$, kde $k \geq q$. Toto se musí se dát rozdělit na pět částí u, v, w, x, y tak, aby $vx \neq \varepsilon$ a $|vwx| \leq q$. Pro žádné z možných rozdělení však neplatí $uv^2wx^2y \in (L_{A,B} \cap S)$, což je spor.

Jazyk $L_{A,B} \cap S$ je tedy bezkontextový jedině tehdy, když je prázdný. $\square$

Jako přímý důsledek právě uvedených tří lemmat dostáváme toto tvrzení:

Věta 5.33. *Pro libovolnou danou bezkontextovou gramatiku $\mathcal{G}$ a libovolnou danou regulární množinu R je nerozhodnutelné určit, zda*

(a) $L(\mathcal{G}) = R$

(b) $L(\mathcal{G}) \supseteq R$

Důkaz. **(a)** Stačí za R zvolit $(\Sigma \cup \{\sharp\} \cup \{1, \ldots, n\})^*$ a za $\mathcal{G}$ bezkontextovou gramatiku generující jazyk co–$(L_{A,B} \cap S)$. Podle lemmatu 5.30 je co–$(L_{A,B} \cap S) = R$ právě když instance $\langle A, B\rangle$ PKP nemá řešení.

(b) Ukážeme, že inkluze $L(\mathcal{G}) \subseteq R$ je snadno rozhodnutelná. Kdyby byla rozhodnutelná i inkluze uvedená v (b), pak bychom uměli rozhodovat i rovnost, což je spor s bodem (a). K rozhodnutelnosti inkluze $L(\mathcal{G}) \subseteq R$ si stačí uvědomit, že $L(\mathcal{G}) \subseteq R \iff L(\mathcal{G}) \cap$ co–$R = \emptyset$ a jazyk $L(\mathcal{G}) \cap$ co–R je bezkontextový.

$\square$

Důsledek 5.34. ***(a)*** *Pro libovolnou danou bezkontextovou gramatiku $\mathcal{G}$ s terminální abecedou Σ je nerozhodnutelné určit, zda $L(\mathcal{G}) = \Sigma^*$.*

(b) *Pro dvě libovolné dané bezkontextové gramatiky $\mathcal{G}_1$ a $\mathcal{G}_1$ je nerozhodnutelné určit, zda $L(\mathcal{G}_1) = L(\mathcal{G}_2)$ resp. zda $L(\mathcal{G}_1) \supseteq L(\mathcal{G}_2)$.*

Věta 5.35. *Pro dvě libovolné dané bezkontextové gramatiky $\mathcal{G}_1$ a $\mathcal{G}_2$ je nerozhodnutelné určit, zda*

(a) *$L(\mathcal{G}_1) \cap L(\mathcal{G}_2)$ je bezkontextový jazyk,*

(b) *co–$L(\mathcal{G}_1)$ je bezkontextový jazyk,*

(c) *$L(\mathcal{G}_1)$ je regulární jazyk* (či ekvivaletně: *jazyk $L(\mathcal{G}_1)$ nemá vlastnost sebevložení).*

Důkaz. **(a)** Stačí zvolit bezkontextové gramatiky $\mathcal{G}_1$ a $\mathcal{G}_2$ tak, aby $L(\mathcal{G}_1) = L_{A,B}$ a $L(\mathcal{G}_2) = S$ a aplikovat lemma 5.32.

(b) Zvolme bezkontextovou gramatiku $\mathcal{G}_1$ tak, aby $L(\mathcal{G}_1) =$ co–$(L_{A,B} \cap S)$. Tvrzení je pak důsledkem lemmatu 5.32.

(c) Provedeme volbu jako v předcházejícím případě. Jazyk $L(\mathcal{G}_1)$ je regulární, právě když je roven $(\Sigma \cup \{\sharp\} \cup \{1, \ldots, n\})^*$. Jazyk co–$(L_{A,B} \cap S)$ je regulární, právě když $L_{A,B} \cap S$ je regulární, a to je (podle lemmatu 5.32) tehdy a jen tehdy, když instance $\langle A, B\rangle$ PKP nemá řešení.

□

Věta 5.36. *Pro libovolnou danou bezkontextovou gramatiku $\mathcal{G}$ je nerozhodnutelné určit, zda je víceznačná.*

Důkaz. Nechť $\langle A, B\rangle$ je libovolná instance Postova korespondenčního problému nad Σ, $A = (x_1, \ldots, x_n)$, $B = (y_1, \ldots, y_n)$. Bez újmy na obecnosti můžeme předpokládat, že $\Sigma \cap \{1, \ldots, n\} = \emptyset$ a $\sharp \notin \Sigma$. K této instanci nyní sestrojíme bezkontextovou gramatiku $\mathcal{G} = (N, \Sigma \cup \{1, \ldots, n\} \cup \{\sharp\}, P, S)$ s množinou pravidel

$$
\begin{array}{llll}
P: & S & \to & S_1 \mid S_2 \\
 & S_1 & \to & x_i S_1 i \mid \sharp \qquad \text{pro všechna } 1 \le i \le n \\
 & S_2 & \to & y_i S_2 i \mid \sharp \qquad \text{pro všechna } 1 \le i \le n
\end{array}
$$

Zřejmě gramatika $\mathcal{G}$ je víceznačná tehdy a jen tehdy, když instance $\langle A, B\rangle$ Postova korespondečního problému má řešení. □

Přehled rozhodnutelných a nerozhodnutelných problémů týkajících se tříd jazyků Chomského hierarchie je obsažen v následující tabulce[5]. Symbol R (resp. N) značí, že problém je rozhodnutelný (resp. nerozhodnutelný). V případě, že vlastnost je splněna pro všechny instance problému, je na odpovídajícím místě v tabulce ano.

	R	*DCF*	*CF*	*CS*	*Rec*	*RE*
Je $L(\mathcal{G})$ prázdný? konečný?	R	R	R	N	N	N
Je $L(\mathcal{G}) = \Sigma^*$?	R	R	N	N	N	N
Je $L(\mathcal{G}) = R$? (R je regulární množina)	R	R	N	N	N	N
Je $L(\mathcal{G}_1) = L(\mathcal{G}_2)$?	R	R	N	N	N	N
Je $L(\mathcal{G}_1) \supseteq L(\mathcal{G}_2)$?	R	N	N	N	N	N
Je $L(\mathcal{G})$ regulární jazyk?	ano	R	N	N	N	N
Je průnik dvou jazyků jazyk téhož typu?	ano	N	N	ano	ano	ano
Je sjednocení dvou jazyků jazyk téhož typu?	ano	N	ano	ano	ano	ano
Je komplement jazyka jazyk téhož typu?	ano	ano	N	ano	ano	N
Je zřetězení dvou jazyků jazyk téhož typu?	ano	N	ano	ano	ano	ano
Je gramatika $\mathcal{G}$ víceznačná?	R	N	N	N	N	N

5. Ne všechny výsledky jsou dokázány v tomto učebním textu.