RNDr. Michal Chytil, CSc.

AUTOMATY A GRAMATIKY

matematický seminář SNTL

PRAHA 1984

Kniha seznamuje čtenáře se základy teorie automatů a formálních jazyků v míře potřebné k pochopení moderních monografií a článků z teorie programování, překladačů programovacích jazyků a operačních systémů.

Je určena zejména systémovým programátorům, posluchačům a učitelům vysokých škol z oborů vychovávajících odborníky zaměřené na používání výpočetní techniky. Může posloužit i učitelům a studentům gymnázií s výukou programování.

Redakce teoretické literatury –
hlavní redaktorka RNDr. Blanka Kutinová, CSc.

OBSAH

PŘEDMLUVA

Problematika automatů a formálních jazyků, o nichž pojednává tato kniha, patří mezi nejdéle studované a nejpodrobněji zpracované disciplíny tvořící obor matematické informatiky. Dosažené výsledky se uplatňují nejen při návrhu počítačů a jejich programového vybavení, ale např. i při studiu živých organismů či přirozených jazyků.

Při psaní knihy jsem měl na mysli čtenáře, který chce teorii automatů a formálních jazyků porozumět do té míry, aby mohl její výsledky a metody užitečně aplikovat. Snažil jsem se vyjít vstříc i těm čtenářům, kteří nejsou školenými matematiky. Proto jsou všechny kapitoly děleny na dvě části, část základní a část rozšiřující.

Základní části nikde neodkazují na části rozšiřující. Je proto možné přečíst si z celé knihy pouze základní části, které tvoří jakýsi úvodní kurs. Při jejich psaní mi pomáhaly zkušenosti z přednášek v několika běžích postgraduálního kursu teoretické kybernetiky na Matematicko-fyzikální fakultě Karlovy univerzity, jejichž posluchači měli velmi různorodé předběžné matematické vzdělání. Matematický formalismus v základních částech je zaváděn zvolna a podrobně komentován, zejména u míst, jejichž pochopení činí podle mých zkušeností nematematikům zprvu potíže. Pro profesionální matematiky budou ovšem příslušné partie poněkud zdlouhavé.

Rozšiřující části kapitol předpokládají o něco vyšší matematickou úroveň nebo průpravu získanou prostudováním základních částí knihy.

Celkový rozsah knihy zhruba odpovídá látce, kterou jsem po několik let přednášel posluchačům oboru matematická informatika a teoretická kybernetika na MFF KU. Zde je látka poněkud modifikována tak, aby se co nejméně překrývala s obsahem monografie Hopcrofta a Ullmana [20], jejíž slovenský překlad vyšel v r. 1978.

Obsah knihy je seskupen okolo dvou hlavních témat – konečných automatů a bezkontextových jazyků. Obě témata mi připadají vhodná k ilustraci výstavby matematické teorie vedoucí k prakticky užitečným výsledkům. V případě konečných automatů jsou uvedeny teoreticky poměrně jednoduché postupy, které umožňují výrazně usnadnit návrh konečných automatů. Části týkající se bezkontextových jazyků nabízejí ukázku výstavby složitějšího teoretického aparátu, který umožňuje pochopit podstatu analýzy LL a LR jazyků, tzn. principů užívaných v soudobých překladačích programovacích jazyků.

První z obou témat by mělo navíc čtenáři ukázat, jak nalézt za matematickým formalismem důležité a zajímavé myšlenky, které lze do jisté míry vyjádřit i neformálně. Druhé téma je naopak voleno tak, aby ukázalo, že pokročilejší techniky je možné opřít pouze o precizní formální vyjádření původních názorných pojmů a úvah.

Vedle zmíněných dvou témat jsou zařazeny i partie poskytující širší pohled na pojednávanou tématiku. Jsou zpravidla probírány volněji a zkratkovitěji.

Tuto knihu lze přečíst, jak obětavě ukázali oba recenzenti, doc. RNDr. Jiří Hořejš, CSc. a RNDr. Miron Tegze. Jsem jim oběma vděčný za desítky rad a připomínek, které mi umožnily opravit celou řadu nedostatků. Pokud jsem někde nedokázal jejich cenné pomoci plně využít, přijde na to bohužel asi sám čtenář, až se mu některá partie bude obtížněji číst.

Přál bych si proto, aby každý z čtenářů byl při čtení nadán aspoň malou částí té trpělivosti, kterou se mnou při psaní této knihy měla má žena Ludmila.

Autor

V dalším výkladu budeme definovat řadu pojmů z teorie automatů a gramatik, budeme však pracovat také s pojmy, které tvoří společný základ soudobých matematických disciplín a s pojmy z teorie grafů.

Množiny. Tohoto pojmu užíváme v běžném intuitivním smyslu. Množiny budeme zpravidla označovat velkými písmeny. Symbolem $x \in X$ značíme, že x *je prvkem množiny* X, $x \notin X$ označuje, že x *není prvkem* X. Vztah *inkluze* mezi dvěma množinami, označovaný symbolem $X \subseteq Y$, znamená, že každý prvek množiny X je zároveň prvkem množiny Y. Říkáme také, že X je *podmnožinou* množiny Y. Množina X je *vlastní podmnožinou* množiny Y (značíme $X \subset Y$ nebo $X \subsetneq Y$), jestliže $X \subseteq Y$ a zároveň $X \neq Y$.

Množiny budeme zpravidla zadávat vlastností, která charakterizuje právě všechny její prvky. Zadání množiny X jakožto množiny všech prvků, které splňují podmínku P, mívá tvar

$$X = \{x; P(x)\}.$$

Jako příklad si uveďme definici několika důležitých operací nad množinami.

Sjednocení

$$A \cup B = \{x; x \in A \text{ nebo } x \in B\},$$

průnik

$$A \cap B = \{x; x \in A \text{ a } x \in B\},$$

rozdíl

$$A - B = \{x; x \in A \text{ a } x \notin B\},$$

kartézský součin

$$A \times B = \{(x, y); x \in A \text{ a } y \in B\}.$$

Sjednocení dvou množin A a B je tedy tvořeno právě prvky, které patří buď do A, nebo do B, nebo do obou množin zároveň, kartézský součin $A \times B$ těchto dvou množin sestává právě z těch uspořádaných dvojic (x, y), jejichž první prvek patří do A a druhý do B atd.

Konečné množiny je možné zadat výčtem prvků. Například zápis $\{a, b, c\}$ označuje množinu tvořenou právě třemi prvky a, b, c. *Prázdnou množinu* budeme označovat symbolem $\emptyset$. Budeme říkat, že dvě množiny A, B jsou *disjunktní*, jestliže nemají společný prvek, tzn. $A \cap B = \emptyset$. *Potenční množinou* $\mathscr{P}(A)$ množiny A nazýváme množinu všech podmnožin množiny A, tzn.

$$\mathscr{P}(A) = \{B; B \subseteq A\}.$$

Logické spojky, kvantifikátory. Pro zpřehlednění komplikovanějších výroků budeme někdy užívat symbolů $\vee$, $\&$, $\Rightarrow$, $\Leftrightarrow$ označujících logické spojky. Jestliže φ a ψ symbolizují výroky, potom

konjunkce těchto dvou výroků, symbolicky $\varphi \,\&\, \psi$, označuje výrok, který říká, že současně platí φ i ψ,

disjunkce $\varphi \vee \psi$ symbolizuje výrok „φ nebo ψ (nebo obojí)“,

implikace $\varphi \Rightarrow \psi$ symbolizuje výrok „z φ plyne ψ“ a

ekvivalence $\varphi \Leftrightarrow \psi$ symbolizuje výrok „φ platí, právě když platí ψ“.

Například formule

$$A = B \Leftrightarrow [A \subseteq B \,\&\, B \subseteq A],$$

zachycuje jednoduchý, ale důležitý fakt, že dvě množiny se rovnají, právě když jedna je součástí druhé a naopak. (Tohoto faktu budeme v dalším textu často používat, budeme-li postaveni před úkol dokázat, že dvě množiny jsou si rovny. Zcela analogicky při

důkazu, že dva výroky jsou ekvivalentní, budeme vycházet z faktu, že pro libovolné dva výroky φ, ψ je

$$[\varphi \Leftrightarrow \psi] \Leftrightarrow [(\varphi \Rightarrow \psi) \,\&\, (\psi \Rightarrow \varphi)],$$

tzn. φ a ψ jsou ekvivalentní, právě když z φ plyne ψ a obráceně.)

Symbol $\forall$ bude označovat *obecný kvantifikátor*, symbol $\exists$ *kvantifikátor existenční*. Zápis $\forall x\varphi$ tedy symbolizuje výrok „pro všechna x platí φ“, naproti tomu zápis $\exists x\varphi$ symbolizuje výrok „existuje x, pro něž platí φ“. Například

$$X \subseteq Y \Leftrightarrow \forall x(x \in X \Rightarrow x \in Y),$$

$$X \subsetneq Y \Leftrightarrow [\forall x(x \in X \Rightarrow x \in Y) \,\&\, \exists x(x \in Y \,\&\, x \notin X)].$$

Relace. *Relací* mezi prvky množiny A a prvky množiny B budeme nazývat každou množinu $R \subseteq A \times B$. Jak je vidět, vycházíme z pohledu, že relace R je určena všemi uspořádanými dvojicemi (x, y) takovými, že $x \in A$, $y \in B$ a x je v relaci R s y. Jestliže x je v relaci R s y, píšeme místo $(x, y) \in R$ častěji xRy.

V případě, že $R \subseteq A \times A$, říkáme, že R je relací na množině A. Taková relace je navíc

reflexívní, jestliže pro všechna $x \in A$ je xRx,

symetrická, jestliže pro všechna $x, y \in A$ je $xRy \Leftrightarrow yRx$,

antisymetrická, jestliže z xRy a yRx plyne $x = y$,

tranzitivní, jestliže z xRy a yRz plyne xRz.

Pro libovolnou relaci R na množině A budeme *reflexívním a tranzitivním uzávěrem relace* R nazývat nejmenší reflexívní a tranzitivní relaci S na A takovou, že $R \subseteq S$. Jinými slovy, S vznikne z R přidáním pouze těch dvojic, které jsou nutné pro to, aby S byla reflexívní a tranzitivní. Tuto relaci můžeme charakterizovat také takto:

$$S = R \cup \{(x, x); x \in A\} \cup$$
$$\cup \{(x, y); (\exists x_1, \ldots, x_n \in A)[xRx_1 \,\&$$
$$\& \, x_1Rx_2 \,\& \ldots \& \, x_nRy]\}.$$

Relaci R na množině A budeme nazývat *částečným uspořádáním*, jestliže je tranzitivní a antisymetrická.

Relace R na množině A je relací *ekvivalence*, jestliže je reflexívní, symetrická a tranzitivní. Pro relace ekvivalence budeme většinou užívat symbol $\sim$ či $\equiv$. Každá relace ekvivalence $\sim$ určuje tzv. třídy ekvivalence. *Třídu ekvivalence* tvoří vždy všechny spolu ekvivalentní prvky, tzn. množina A' je třídou ekvivalence $\sim$ na množině A, jestliže

1. pro každé $x, y \in A'$ je $x \sim y$ a
2. jestliže $x \in A'$ a $x \sim y$, potom $y \in A'$.

Třídu ekvivalence $\sim$ obsahující prvek x značíme $[x]_{\sim}$ nebo jen $[x]$, je-li z kontextu jasné, o kterou relaci ekvivalence se jedná.

Jestliže $A_1, A_2, \ldots, A_n$ jsou všechny třídy ekvivalence $\sim$ na množině A, potom tyto třídy tvoří *rozklad* A; tzn. každé dvě různé třídy jsou disjunktní a všechny dohromady pokrývají A, tj.

1. $(\forall i, j \in \{1, \ldots, n\})[i \neq j \Rightarrow A_i \cap A_j = \emptyset]$,
2. $A_1 \cup A_2 \cup \ldots \cup A_n = A$.

Množina všech tříd rozkladu $\{A_1, \ldots, A_n\}$ se nazývá *podílová množina* množiny A podle $\sim$ a značí se $A/\sim$.

Zobrazení. Speciálním případem relace je zobrazení. *Zobrazení* z množiny A do množiny B je každá relace $F \subseteq A \times B$ taková, že pro každé $x \in A$ existuje nejvýše jedno $y \in B$ tak, že xFy. Místo xFy bývá v případě zobrazení zvykem psát $F(x) = y$. Prvek y pak nazýváme *hodnotou F v bodě x* nebo *obrazem prvku x* při zobrazení F. Obráceně, prvek x se nazývá *vzor prvku y* při zobrazení F. Skutečnost, že F je zobrazení z množiny A do množiny B, zachycujeme symbolem $F: A \to B$.

Chceme-li zdůraznit, že pro některá $x \in A$ nemusí být $F(x)$ definováno (tzn. pro žádné $y \in B$ není xFy), mluvíme o F jako o *parciálním zobrazení*. Pokud je pro každé $x \in A$ definováno $F(x)$, říkáme, že F je *totální zobrazení*. Zobrazení $F: A \to B$ takové, že pro každé dva různé prvky x, y z A je $F(x) \neq F(y)$, se nazývá *prosté*. Jestliže je navíc zobrazením *na* B, tj. pro každé $y \in B$ existuje $x \in A$ tak, že $F(x) = y$, nazývá se F *bijekce*.

Symbolem id_A budeme označovat *identické zobrazení*

$$\mathrm{id}_A: A \to A,$$

tj. zobrazení takové, že

$$\mathrm{id}_A(x) = x$$

pro každé $x \in A$.

Grafy. V části týkající se formálních jazyků budeme užívat několik základních pojmů z teorie grafů. (*Orientovaným*) *grafem* budeme nazývat dvojici (V, H), kde V je konečná neprázdná množina (*množina vrcholů*) a $H \subseteq V \times V$ (*množina hran*). O hraně $(u, v) \in H$ pak říkáme, že *vychází z vrcholu* u a *vchází do vrcholu* v. Konečná posloupnost hran $(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)$ se nazývá *cesta délky* n z u_1 do v_n, jestliže $v_i = u_{i+1}$ pro $i = 1, \ldots, n - 1$.

Graf (V, H) nazýváme *stromem*, jestliže

1. existuje právě jeden vrchol, do něhož nevchází žádná hrana (takový vrchol se nazývá *kořen stromu*) a

2. pro každý vrchol v různý od kořene existuje právě jedna cesta z kořene do v.

Každý strom samozřejmě obsahuje vrcholy, z nichž nevychází žádná hrana. Takové vrcholy nazýváme *listy stromu*. Ostatní vrcholy nazýváme *vnitřními vrcholy*. *Výšku stromu* definujeme jako délku nejdelší cesty z kořene do některého listu. Výšku *triviálního stromu* tvořeného jediným vrcholem pokládáme za rovnou nule.

1. KONEČNÉ AUTOMATY A JAZYKY ROZPOZNATELNÉ KONEČNÝMI AUTOMATY

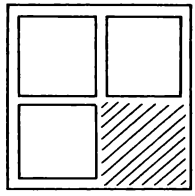
ZÁKLADNÍ ČÁST

1.1. Konečné automaty a jejich reprezentace

Předmětem zájmu teorie konečných automatů je každý systém, u kterého můžeme vymezit konečně mnoho stavů, do nichž se může dostat, a konečně mnoho druhů vnějších podnětů působících změny stavů. Přitom je podstatné, aby stav systému a vnější podnět vždy dohromady jednoznačně určovaly následující stav. Od fyzikální povahy systému, jeho geometrické struktury apod. zcela abstrahujme.

Pro ilustraci uveďme tři velmi jednoduché příklady takových systémů.

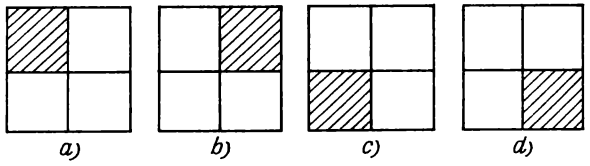
Příklad 1.1.1. Mějme krabičku se čtvercovým dnem o hraně délky d a v ní tři kostky o hranách délky $d/2$ (viz obr. 1; prázdný prostor je vyšrafován).



Obr. 1

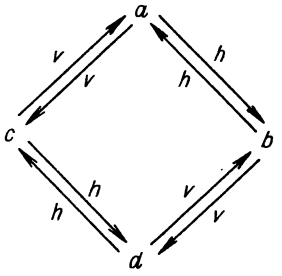
Omezíme-li se na situace, kdy jsou kostky umístěny v rozích krabičky, dostáváme pouze čtyři navzájem různé stavy, tak jak jsou zachyceny na obr. 2. Přecházet z jednoho stavu do druhého je možné posunem některé kostky na volný čtverec. V každém ze stavů lze posunout právě jednu kostku horizontálním nebo vertikálním směrem. Označme horizontální posun písmenem h a vertikální písmenem v. Přechody mezi stavy označenými na obr. 2 písmeny a, b, c, d jsou znázorněny obr. 3.

Šipky označené písmenem h označují změny stavů způsobené horizontálními posuny, šipky v odpovídají vertikálním posunům. Úroveň abstrakce, se kterou je výchozí systém zachycen na obr. 3, je pro teorii automatů charakteristická.

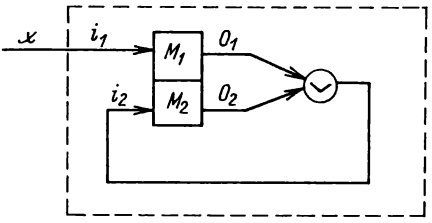


Obr. 2

Příklad 1.1.2. Obrázek 4 představuje jednoduchý sekvenční obvod. M_1 a M_2 jsou dvě paměti, z nichž každá může uchovávat buď hodnotu 0, nebo 1. Paměti pracují v diskrétní časové škále. V každém časovém okamžiku vystupuje obsah pamětí na výstupech o_1, o_2 a jsou do nich ukládány symboly ze vstupů i_1 a i_2.



Obr. 3

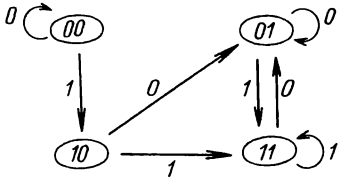


Obr. 4

Výstupy o_1, o_2 jsou přiváděny do hradla realizujícího logickou disjunkci a výstup tohoto hradla je připojen na vstup i_2. To znamená, že do paměti M_2 vstupuje jednička právě tehdy, jestliže ji v předchozím okamžiku obsahovala alespoň jedna z obou pamětí. Na vstup i_1 se v každém okamžiku přivádí jedna ze vstupních hodnot, tzn. 0 nebo 1.

Vstupními symboly tohoto automatu tedy budou symboly 0 a 1 a celé zařízení se může dostávat do čtyř různých stavů podle toho, jaký je stav pamětí M_1 a M_2. Tyto stavy je možno charakterizovat dvojicemi 00, 01, 10, 11. První symbol ve dvojici za-

chycuje obsah paměti M_1, druhý symbol obsah paměti M_2. Přechody mezi stavy automatu, způsobené vstupními hodnotami 0 a 1, lze znázornit diagramem na obr. 5.



Obr. 5

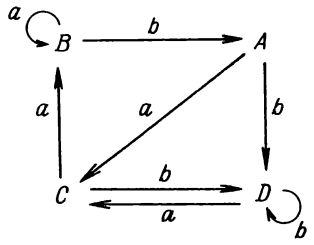
V obou příkladech bylo nalezení konečné množiny stavů jednoduchou a víceméně jednoznačnou záležitostí. Ukážeme si proto nyní, jak i v komplikovaných a nekonečných systémech lze někdy nalézt strukturu odpovídající konečnému automatu. Kromě toho se bude následující příklad na rozdíl od předchozích vyznačovat naprostou nezávislostí na konkrétní realizaci.

Příklad 1.1.3. Zkoumejme systém psaní slov složených ze symbolů *a*, *b*. Elementárním stavem tohoto systému je vždy nějaká konečná posloupnost (slovo) ze symbolů *a*, *b* a dvěma vnějšími podněty měnícími stav systému je připisování symbolů *a* či *b* zprava. Všech možných konečných posloupností symbolů *a*, *b* je nekonečně mnoho, a proto i elementárních stavů tohoto systému je nekonečně mnoho. Přesto lze rozdělit všechny uvažované posloupnosti do několika disjunktních tříd tak, že při tomto hrubším pohledu se celý systém bude jevit jako konečný automat. Uveďme jedno z možných rozdělení na třídy:

A obsahuje všechny posloupnosti končící na *aab*,
B obsahuje všechny posloupnosti končící na *aa*,
C obsahuje všechny posloupnosti končící na *a* a nepatřící do *B* (tzn. patří sem posloupnosti končící na *ba* a jednočlenná posloupnost *a*),
D obsahuje všechny posloupnosti nepatřící do $A \cup B \cup C$.

Je vidět, že libovolná posloupnost patřící do třídy *A* po připsání symbolu *a* zprava přejde do *C*, zatímco po připsání *b* každá taková posloupnost přejde do *D*. Podobně libovolná posloupnost

z třídy B zůstane po připsání symbolu a opět v B, zatímco připsání b ji převede do A. Analogicky pro třídy C a D, jak to znázorňuje obr. 6.



Obr. 6

Uvedený rozklad množiny posloupností na čtyři třídy je jen jednou z nekonečně mnoha možností, jak na uvedený systém pohlížet jako na konečný automat. Zkuste si sami načrtnout podobný diagram jako je na obr. 6, např. pro rozklad na pouhé dvě třídy, z nichž jedna obsahuje právě ty posloupnosti, které obsahují sudý počet výskytů symbolu a a druhá právě posloupnosti s lichým počtem výskytů a.

K charakteristikám konečného automatu, které jsme neformálně vyslovili v úvodu, přistupuje ještě určení jednoho stavu jako počátečního a množiny koncových stavů.

Definice 1.1.4. *Konečným automatem* nazýváme každou pětici $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$, kde Q je konečná neprázdná množina (*množina stavů, stavový prostor*), Σ je konečná neprázdná množina (*množina vstupních symbolů, vstupní abeceda*), δ je zobrazení $Q \times \Sigma \to Q$ (*přechodová funkce*), $q_0 \in Q$ (*počáteční stav, iniciální stav*), $F \subseteq Q$ (*cílová množina, množina koncových stavů*).

Konečný automat lze specifikovat zadáním všech pěti parametrů, které jej určují.

Příklad 1.1.5. Jistý konečný automat $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$ je zadán výčtem stavů $Q = \{q_0, q_1, q_2, q_3\}$, vstupních symbolů $\Sigma = \{0, 1\}$, hodnotami přechodové funkce pro jednotlivé argumenty

$$\delta(q_0, 0) = q_0\,, \qquad \delta(q_0, 1) = q_2\,,$$
$$\delta(q_1, 0) = q_1\,, \qquad \delta(q_1, 1) = q_3\,,$$

$$\delta(q_2, 0) = q_1, \qquad \delta(q_2, 1) = q_3,$$
$$\delta(q_3, 0) = q_1, \qquad \delta(q_3, 1) = q_3,$$

specifikováním počátečního stavu q_0 a cílové množiny $F = \{q_1, q_2\}$.

Lepší představě o struktuře konečného automatu napomáhá vhodná grafická reprezentace automatu. V dalším textu budeme používat tři typy reprezentace konečného automatu. Budeme je nyní ilustrovat na automatu z předchozího příkladu.

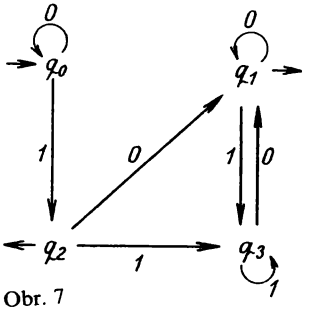
Konečný automat můžeme reprezentovat tabulkou, která v případě automatu z př. 1.1.5 vypadá takto:

	0	1
$\rightarrow q_0$	q_0	q_2
$\leftarrow q_1$	q_1	q_3
$\leftarrow q_2$	q_1	q_3
q_3	q_1	q_3

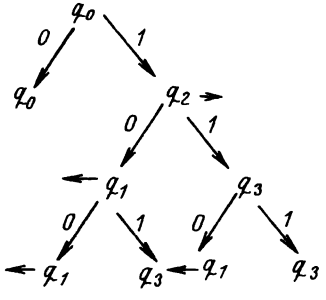
Tabulka udává přechodovou funkci δ automatu, počáteční stav je v ní označen $\rightarrow$ (šipkou mířící na symbol stavu), koncové stavy $\leftarrow$ (šipkou mířící ze symbolu stavu) a v případě, že počáteční stav leží v cílové množině, je označen $\leftrightarrow$.

Dalším typem reprezentace je stavový diagram (obr. 7):

Vrcholy stavového diagramu odpovídají stavům automatu, hrany označují možné přechody mezi stavy a jsou ohodnoceny příslušnými vstupními symboly. Obdobně jako v tabulce jsou po-



Obr. 7



Obr. 8

čáteční stav, resp. koncové stavy označeny neohodnocenými šipkami do stavu vcházejícími, resp. z něho vycházejícími. (Až na označení význačných stavů jsou tedy obr. 5 a 6 stavové diagramy.)

Nakonec uvedeme reprezentaci stavovým stromem (obr. 8):

Ve stavovém stromu z každého vrcholu, který není listem, vychází právě tolik hran, kolik je symbolů vstupní abecedy. Kořen stromu je ohodnocen počátečním stavem. Následníci každého vrcholu jsou ohodnoceni v souladu s přechodovou funkcí. Jestliže se nějaký stav objeví jako ohodnocení několika vrcholů, stačí, když hrany vycházejí jen z jednoho z těchto vrcholů. Koncové stavy jsou vyznačeny stejně jako v předchozích případech, počáteční stav není třeba vyznačovat.

Stavovým stromem je možné reprezentovat pouze automaty, u nichž je každý stav dosažitelný z počátečního stavu. Pochopitelně nic podstatného neztrácíme, jestliže se omezíme pouze na takové automaty (podrobněji viz lemma 1.5.3).

Stavový diagram dostaneme ze stavového stromu tak, že ztotožníme vrcholy ohodnocené stejným stavem a vyznačíme počáteční stav.

Každá z uvedených reprezentací automatu umožňuje snadno zjistit, jaká bude reakce automatu na libovolnou posloupnost vstupních symbolů.

Tak automat z př. 1.1.5 reaguje na posloupnost symbolů 010010 tak, že se z počátečního stavu q_0 dostane postupně do stavů q_0, q_2, q_1, q_1, q_3, q_1. Uvedená posloupnost tedy převede automat ze stavu q_0 do stavu q_1. Posloupností s touto vlastností je zřejmě nekonečně mnoho. Při pozornějším prozkoumání automatu zjistíme, že jsou to právě všechny posloupnosti obsahující alespoň jednu jedničku a končící nulou. V dalším výkladu bude středem naší pozornosti právě možnost charakterizovat množiny (i nekonečné) vstupních posloupností pomocí konečných automatů.

Vstupní posloupnosti lze interpretovat různými způsoby. Nejvíce se nabízí možnost chápat každou posloupnost vstupních symbolů jako sled obrazů jisté probíhající události tak, jak je zachycena „čidly" automatu. Některé události převedou automat z počátečního stavu do koncového stavu. Pokud by automat po-

pisoval kontrolní zařízení, pak by skutečnost, že se dostal do koncového stavu, mohla například znamenat, že zaznamenal kombinaci vstupních symbolů signalizujících poruchu.

Jiná možnost je chápat posloupnosti vstupních symbolů jako řetěz symbolů v jisté abecedě, čili jako jakési věty. Tuto představu budeme formalizovat v následujícím článku.

1.2. Jazyky rozpoznatelné konečnými automaty

Definice 1.2.1. Je-li Σ libovolná konečná množina (abeceda), pak Σ^+ označuje množinu všech konečných neprázdných posloupností utvořených z prvků množiny Σ, e označuje prázdnou posloupnost a definujeme $\Sigma^* = \Sigma^+ \cup \{e\}$.

Posloupnost $a_1, \ldots, a_n \in \Sigma^*$ budeme zapisovat $a_1 \ldots a_n$. Každou takovou posloupnost nazýváme *řetězem* nebo *slovem v abecedě* Σ, e nazýváme *prázdným slovem.*

O Σ^* mluvíme jako o *množině všech řetězů nad abecedou* Σ, o Σ^+ jako o *množině neprázdných řetězů nad* Σ.

Jestliže $u = a_1 \ldots a_n$ a $v = b_1 \ldots b_m \in \Sigma^*$, pak řetěz $uv = a_1 \ldots a_n b_1 \ldots b_m$ nazveme *zřetězením slov* u a v. Speciálně $eu = ue = u$. Symbol u^n bude označovat *n-násobné zřetězení slova* u, tzn. $u^1 = u$, $u^2 = uu$, $u^{i+1} = u^i u$. *Délku řetězu* u budeme značit $|u|$, tj. $|a_1 \ldots a_n| = n$ $(a_i \in \Sigma)$ a $|e| = 0$.

Definice 1.2.2. Je-li Σ konečná abeceda a $L \subseteq \Sigma^*$, pak L nazýváme *jazykem nad abecedou* Σ.

Ve smyslu této definice je každý přirozený jazyk, např. i čeština, jazykem nad konečnou abecedou. Řetězy tohoto jazyka jsou všechny správně utvořené české věty. Do příslušné konečné abecedy patří kromě všech symbolů české abecedy ještě všechna interpunkční znaménka a mezera. Jiným typem objektů, které je užitečné zkoumat aparátem formálních jazyků, jsou programovací jazyky. Tak např. řetězem patřícím do formálního jazyka Pascal je každý syntakticky správný program v tomto jazyce.

Dohodněme se nyní na způsobu, jakým bude každý konečný automat reprezentovat jistý jazyk.

Definice 1.2.3. Přechodovou funkci $\delta: Q \times \Sigma \to Q$ konečného

automatu $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$ rozšíříme na *zobecněnou přechodovou funkci* δ^*: $Q \times \Sigma^* \to Q$ takto:

1. $\delta^*(q, e) = q$ pro každé $q \in Q$,
2. $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$ pro každé $q \in Q$, $w \in \Sigma^*$, $a \in \Sigma$.*)

Jazykem rozpoznávaným konečným automatem $\mathscr{A}$ pak nazveme jazyk

$$L(\mathscr{A}) = \{w;\ w \in \Sigma^* \ \&\ \delta^*(q_0, w) \in F\}.$$

Říkáme, že *řetěz* $w \in \Sigma^*$ *je přijímán automatem* $\mathscr{A}$, právě když $w \in L(\mathscr{A})$.

Definice 1.2.4. Existuje-li konečný automat $\mathscr{A}$ takový, že jazyk $L = L(\mathscr{A})$, říkáme, že *jazyk* L *je rozpoznatelný konečným automatem.*

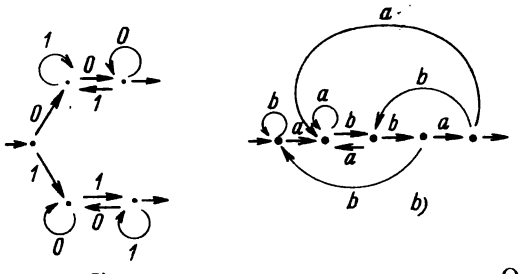
Symbolem $\mathscr{F}$ budeme označovat množinu všech jazyků rozpoznatelných konečným automatem.

Uveďme si dva příklady jazyků patřících do $\mathscr{F}$.

Příklad 1.2.5. Jazyky

$L_1 = \{w \in \{0, 1\}^*;\ w$ začíná i končí stejným symbolem, $|w| \geqq 2\}$,
$L_2 = \{w \in \{a, b\}^*;\ w$ končí řetězem $abba\}$

jsou rozpoznávané konečnými automaty, které jsou znázorněné na obr. 9a), b).



Obr. 9

*) Funkce δ^* je rozšířením funkce δ. Nedojde proto k vážnějšímu nedorozumění, jestliže v dalším textu budeme obě tyto funkce označovat společným symbolem δ.

Tento příklad naznačuje způsob, jakým lze ověřit určitý vztah mezi dvěma omezenými úseky daného řetězu, a způsob, jak ověřit, zda zkoumaný řetěz obsahuje daný podřetěz. Vhodnými kombinacemi (se kterými se seznámíme v kap. 2) obou těchto způsobů lze zkonstruovat automaty schopné např. kontrolovat syntaktickou správnost programů v jednoduchých programovacích jazycích asi na úrovni jazyků symbolických adres.

Existují ovšem i poměrně jednoduché jazyky, u kterých na první pohled vzniká pochybnost, zda jsou konečnými automaty rozpoznatelné. Příkladem je jazyk

$$\{0^n 1^n;\ n \geqq 0\},$$

tj. jazyk, jehož řetězy jsou tvořeny libovolně dlouhým úsekem složeným ze samých nul následovaným úsekem ze stejného počtu jedniček. Není zřejmé, jak by si měl konečný automat po přečtení nul „zapamatovat" jejich počet, aby jej mohl porovnat s počtem jedniček. A skutečně tento jazyk není konečným automatem rozpoznatelný. S jednoduchým a elegantním způsobem, jak to dokázat, se seznámíme v následujícím článku.

1.3. Nerodova věta

Věta, kterou v tomto článku dokážeme, se nejčastěji užívá při důkazech, že nějaký jazyk není rozpoznatelný konečným automatem.

Definice 1.3.1. Budiž Σ konečná abeceda a $\sim$ relace ekvivalence na Σ^*. Relace $\sim$ se nazývá *pravou kongruencí*, jestliže pro všechna $u, v, w \in \Sigma^*$ ze vztahu $u \sim v$ plyne, že $uw \sim vw$.

Definice 1.3.2. O relaci ekvivalence $\sim$ na množině M říkáme, že je *konečného indexu*, jestliže rozklad $M/\sim$ má konečný počet tříd.

Věta 1.3.3 (Nerodova). *Nechť L je jazyk nad konečnou abecedou Σ. Pak L je rozpoznatelný konečným automatem, právě když existuje pravá kongruence konečného indexu taková, že L je sjednocením jistých tříd rozkladu $\Sigma^*/\sim$.*

Důkaz. 1. Nechť L je jazyk rozpoznatelný konečným automatem $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$. Definujme relaci ekvivalence $\sim$ předpisem

$$u \sim v \Leftrightarrow_{\text{df}} \delta(q_0, u) = \delta(q_0, v) \text{ pro libovolná } u, v \in \Sigma^* .$$

Relace $\sim$ je zřejmě konečného indexu, neboť počet tříd ekvivalence je nejvýše roven počtu stavů automatu $\mathscr{A}$ (přesněji – je roven počtu stavů, do kterých je možné přejít z počátečního stavu q_0 na základě nějaké vstupní posloupnosti). Relace $\sim$ je také pravou kongruencí, neboť z $\delta(q_0, u) = \delta(q_0, v)$ plyne, že pro libovolné $w \in \Sigma^*$ platí $\delta(q_0, uw) = \delta(q_0, vw)$. Protože

$$L = \{w \in \Sigma^*; \delta(q_0, w) \in F\} = \bigcup_{q \in F} \{w; \delta(q_0, w) = q\},$$

je L sjednocením jistých tříd rozkladu $\Sigma^*/\sim$.

2. Nechť relace $\sim$ je pravá kongruence konečného indexu na Σ^* a nechť existují třídy rozkladu $c_1, \ldots, c_n \in \Sigma^*/\sim$ takové, že L je jejich sjednocením, tj.

$$L = c_1 \cup c_2 \cup \ldots \cup c_n = \bigcup_{i=1}^{n} c_i .$$

Dohodněme se, že třídu rozkladu $\Sigma^*/\sim$ obsahující slovo u budeme označovat $[u]$. Pak definujme automat

$$\mathscr{A}' = (Q', \Sigma, \delta', q_0', F')$$

takto:

$$Q' = \Sigma^*/\sim , \qquad q_0' = [e], \qquad F' = \{c_1, \ldots, c_n\}$$

a δ' je zadána předpisem

$$(*) \qquad \delta'([w], a) = [wa] \quad \text{pro každé } w \in \Sigma^*, \ a \in \Sigma .$$

Je třeba se ještě přesvědčit, že automat je definován korektně. Podezření vzbuzuje definice přechodové funkce δ', která se opírá o určení třídy ekvivalence pomocí reprezentanta. Kdyby bylo možné najít slova u a v taková, že $\delta'([u], a) \neq \delta'([v], a)$ pro nějaké $a \in \Sigma$, $[u] = [v]$, potom by hodnota funkce $\delta'([u], a)$ závisela na volbě reprezentanta třídy ekvivalence a předpis $(*)$ by

nebyl korektní definicí funkce. Tato potíž ale nemůže nastat, protože $\sim$ je pravou kongruencí. Jestliže totiž $[u] = [v]$, znamená to, že $u \sim v$, a tedy pro každé $a \in \Sigma$ je také $ua \sim va$, tj. $[ua] = [va]$. Proto také $\delta'([u], a) = \delta'([v], a)$. To znamená, že $\mathscr{A}'$ je korektně definovaný automat. $\mathscr{A}'$ je konečný automat, protože $\Sigma^*/\sim$ je podle předpokladu konečná množina ($\sim$ je konečného indexu).

Zbývá ověřit, že automat $\mathscr{A}'$ rozpoznává jazyk L:

$$\begin{aligned} w \in L &\Leftrightarrow w \in c_1 \vee w \in c_2 \vee \ldots \vee w \in c_n \Leftrightarrow \\ &\Leftrightarrow [w] = c_1 \vee [w] = c_2 \vee \ldots \vee [w] = c_n \Leftrightarrow \\ &\Leftrightarrow \delta'([e], w) = c_1 \vee \delta'([e], w) = \\ &= c_2 \vee \ldots \vee \delta'([e], w) = c_n \Leftrightarrow \\ &\Leftrightarrow \delta'(q_0', w) \in F' . \end{aligned}$$

Obě konstrukce použité v důkazu ozřejmíme na příkladech.

Příklad 1.3.4. Vraťme se k automatu z př. 1.1.5 (viz obr. 7). Tomuto automatu odpovídá pravá kongruence daná rozkladem Σ^* na tyto třídy:

$$T_0 = \{w; \delta(q_0, w) = q_0\}$$

třída slov neobsahujících žádnou jedničku (sem patří i prázdné slovo);

$$T_1 = \{w; \delta(q_0, w) = q_1\}$$

třída slov obsahujících alespoň jednu jedničku a končících na nulu;

$$T_2 = \{w; \delta(q_0, w) = q_2\}$$

třída slov obsahujících jedinou jedničku, která je zároveň posledním symbolem slova;

$$T_3 = \{w; \delta(q_0, w) = q_3\}$$

třída slov končících na jedničku a obsahujících alespoň dvě jedničky.

Jazyk rozpoznávaný automatem je pak sjednocením tříd T_2 a T_1.

Příklad 1.3.5. Vyjděme z rozkladu množiny $\{a, b\}^*$ na třídy A, B, C, D popsaném v př. 1.1.3. Pozorování z tohoto příkladu vedou k závěru, že uvedený rozklad odpovídá pravé kongruenci konečného indexu. Jestliže definujeme jazyk L jako sjednocení tříd A a C, potom tento jazyk je rozpoznáván automatem, jehož přechodová funkce je znázorněna na obr. 6, koncové stavy jsou A, C a počáteční stav je D (třída D obsahuje prázdné slovo).

Typické použití věty 1.3.3 ukazují následující dva příklady.

Příklad 1.3.6. Jazyk $L = \{0^n 1^n; n \geqq 1\}$ není rozpoznatelný konečným automatem.

Důkaz. Provedeme důkaz sporem. Předpokládejme, že L je rozpoznatelný konečným automatem. V tom případě musí existovat pravá kongruence $\sim$ na $\{0, 1\}^*$ konečného indexu taková, že L je sjednocením jistých tříd z $\{0, 1\}^*/\sim$. Nechť $\{0, 1\}^*/\sim$ obsahuje n_0 tříd. Potom alespoň dvě z $n_0 + 1$ slov

$$0, 00, 0^3, 0^4, \ldots, 0^{n_0+1}$$

leží ve stejné třídě. Jinými slovy, $0^i \sim 0^j$ pro nějaká i, j taková, že $1 \leqq i < j \leqq n_0 + 1$. K oběma těmto slovům připíšeme zprava stejné slovo 1^i. Protože $\sim$ je pravá kongruence, dostáváme $0^i 1^i \sim 0^j 1^i$. L je podle předpokladu sjednocením jistých tříd rozkladu, což znamená, že každá třída rozkladu buď celá leží v L, nebo celá leží vně L. Jelikož $0^i 1^i \in L$, mělo by také $0^j 1^i$ ležet v L. Podle definice tohoto jazyka je ale $0^j 1^i \notin L$, což je ve sporu s předpokladem, že L je rozpoznatelný konečným automatem.

Příklad 1.3.7. Jazyk $L = \{a^{n^2}; n \geqq 0\}$ není rozpoznatelný konečným automatem.

Důkaz opět provedeme sporem. Předpokládáme existenci příslušné pravé kongruence $\sim$ o k třídách ekvivalence, takové, že L je sjednocením jistých jejich tříd. Potom alespoň dvě ze slov

$$a^{k^2}, a^{k^2+1}, \ldots, a^{k^2+k}$$

leží ve stejné třídě ekvivalence, tj.

$$a^{k^2+i} \sim a^{k^2+j}$$

pro nějaká i, j taková, že

$$0 \leqq i < j \leqq k .$$

Tedy také

$$a^{k^2+i}a^{2k-j+1} \sim a^{k^2+j}a^{2k-j+1} .$$

Přitom ale

$$a^{k^2+j}a^{2k-j+1} = a^{k^2+2k+1} = a^{(k+1)^2} \in L ,$$

zatímco

$$k^2 < k^2 + i + 2k - j + 1 < (k+1)^2 ,$$

a proto

$$a^{k^2+i+2k-j+1} \notin L .$$

Struktura důkazů v př. 1.3.6 a 1.3.7 je stejná. Liší se pouze volbou řetězů, které slouží jako východisko k odvození sporu. Nalezení takového řetězu se zpravidla opírá o intuitivní náhled, že zkoumaný jazyk má vlastnost, k jejímuž postižení je konečná paměť automatu nedostatečná. Tak v prvním příkladu se nabízí postřeh, že automat nebude schopen zapamatovat si u dostatečně dlouhých slov počet symbolů 0 v první polovině slova, aby jej mohl porovnat s počtem symbolů 1 ve druhé polovině. Jinak řečeno, pro dostatečně velké n nebude automat schopen vzájemně rozlišit všechna slova tvaru $0, 0^2, 0^3, \ldots, 0^n$. Tímto způsobem se dojde k posloupnosti použité v prvním příkladu. Druhý důkaz vychází z postřehu, že rozdíl mezi k^2 a $(k+1)^2$, který je roven $2k + 1$, se s rostoucím k neomezeně zvětšuje.

Uvedené použití Nerodovy věty, i když nejčastější, není jediné. Jinou její zajímavou aplikaci ukážeme v čl. 2.5.

1.4. Kritéria pro návrh konečného automatu

Definice 1.4.1. Dva konečné automaty $\mathscr{A}$, $\mathscr{B}$ se nazývají *ekvivalentní*, jestliže rozpoznávají tentýž jazyk, tzn. $L(\mathscr{A}) = L(\mathscr{B})$.

Ke každému konečnému automatu existuje nekonečně mnoho dalších s ním ekvivalentních automatů. To je vidět např. z toho,

že ke každému automatu lze přidat libovolný počet zbytečných, z počátečního stavu nedosažitelných stavů.

Problém najít k danému jazyku konečný automat, který by jej rozpoznával, tedy buď vůbec nemá řešení, anebo má nekonečně mnoho různých řešení. Při výběru jednoho z těchto řešení se zpravidla explicitně či implicitně řídíme určitým kritériem. Většinou převládá jedno ze dvou základních kritérií.

Prvním z nich je požadavek, aby navržený automat byl co možná nejmenší, tj. aby měl co možná nejmenší počet stavů. Čím menší je totiž počet stavů automatu, tím menší paměť stačí pro jeho technickou realizaci. Podrobněji se touto otázkou budeme zabývat v čl. 1.6.

Druhým kritériem je přehlednost a jasnost návrhu umožňující snadno ověřit jeho správnost.

Obě kritéria jsou často v rozporu. Zmenšení automatu obvykle dosáhneme za cenu, že se jeho struktura stane nepřehlednější.

Ve 2. kapitole, kde se budeme podrobněji věnovat metodám návrhu konečných automatů, budeme výrazně preferovat kritérium přehlednosti a jednoduchosti návrhu. Tento přístup podporuje mj. i existence tzv. algoritmu redukce, kterému je věnován čl. 1.5. Tento algoritmus umožňuje snadno přejít od libovolného konečného automatu navrženého k rozpoznávání určitého jazyka k vůbec nejmenšímu možnému konečnému automatu rozpoznávajícímu tentýž jazyk. Z toho důvodu nebývá nutné věnovat kritériu minimality přílišnou pozornost již ve stadiu návrhu automatu.

ROZŠIŘUJÍCÍ ČÁST

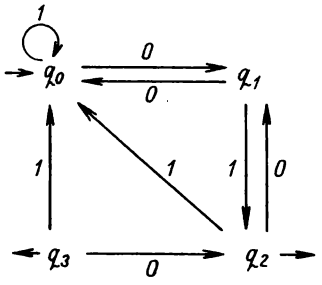
1.5. Redukce konečného automatu

Definice 1.1.4 konečného automatu připouští existenci stavů, do nichž se nelze dostat z počátečního stavu na základě žádné vstupní posloupnosti.

Definice 1.5.1. *Stav* q konečného automatu $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ se nazývá *dosažitelný*, jestliže existuje slovo $w \in \Sigma^*$ takové, že

$\delta(q_0, w) = q$. Stavy, které nejsou dosažitelné, se nazývají *nedosažitelné*.

Příklad 1.5.2. V automatu na obr. 10 jsou stavy q_0, q_1, q_2 dosažitelné a stav q_3 je nedosažitelný.



Obr. 10

Zjistit dosažitelné stavy automatu je snadný úkol. Na začátku označíme počáteční stav a dále postupně značíme všechny stavy, do kterých je možné se dostat z některého již označeného stavu v jediném kroku (tzn. stavy, do nichž ve stavovém diagramu vede z některého již označeného stavu orientovaná hrana). Proceduru zakončíme v okamžiku, kdy už nelze označit žádný další stav. Označené stavy jsou pak právě všechny dosažitelné stavy.

Odstraněním nedosažitelných stavů dostaneme automat ekvivalentní s původním automatem.

Lemma 1.5.3. *Nechť* $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$ *je konečný automat a nechť* $P \subseteq Q$ *je množina dosažitelných stavů automatu* $\mathscr{A}$. *Potom automat* $\mathscr{B} = (P, \Sigma, \delta_p, q_0, F \cap P)$, *kde* δ_p *je parcializací přechodové funkce na* $P \times \Sigma$, *je automat ekvivalentní s* $\mathscr{A}$ *a neobsahuje nedosažitelné stavy.*

Důkaz. Je zřejmé, že stavy, kterými automat projde, než se dostane do nějakého dosažitelného stavu, jsou opět všechny dosažitelné. Proto platí, že

$$(*) \qquad \delta(q_0, w) = \delta_p(q_0, w) \quad \text{pro každé } w \in \Sigma^*.$$

Množina koncových stavů, do kterých se $\mathscr{A}$ může dostat z počátečního stavu, je zřejmě rovna množině $F \cap P$. Pro všechna

$w \in \Sigma^*$ je proto $\delta(q_0, w) \in F \Leftrightarrow \delta_p(q_0, w) \in F \cap P$, tzn. $L(\mathscr{A}) = L(\mathscr{B})$. Ze vztahu (*) také plyne, že $\mathscr{B}$ neobsahuje nedosažitelné stavy.

V každém automatu je dáno základní rozlišení stavů na koncové a ostatní stavy. Dále potom někdy lze dva stavy rozlišit pomocí řetězu vstupních symbolů, který jeden z nich převede do cílové množiny, zatímco druhý stav převede vně této množiny. Stavy, které takto nelze rozlišit žádným slovem, se nazývají ekvivalentní.

Definice 1.5.4. Nechť $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$ je konečný automat a $p, q \in Q$. Řekneme, že *stavy p, q jsou ekvivalentní*, symbolicky $p \sim q$, jestliže pro každé $w \in \Sigma^*$ je

$$\delta(p, w) \in F \Leftrightarrow \delta(q, w) \in F .$$

Relace $\sim$ z předchozí definice je zřejmě relací ekvivalence na Q. Ukážeme jednoduchý způsob, jak pro libovolné dva stavy rozhodnout, zda jsou ekvivalentní.

Definice 1.5.5. Pro každý konečný automat $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$ definujme *posloupnost relací* $\overset{0}{\sim}$, $\overset{1}{\sim}$, $\overset{2}{\sim}$, ... *na množině* Q takto:

$$p \overset{0}{\sim} q \Leftrightarrow_{\text{df}} p \in F \Leftrightarrow q \in F ,$$

tj. oba stavy zároveň leží nebo zároveň neleží v F,

$$p \overset{i}{\sim} q \Leftrightarrow_{\text{df}} p \overset{i-1}{\sim} q \text{ a pro všechna } a \in \Sigma \text{ je}$$

$$\delta(p, a) \overset{i-1}{\sim} \delta(q, a), \quad i \geqq 1 .$$

Je užitečné si uvědomit, že $p \overset{i}{\sim} q$, právě když libovolné slovo délky nepřesahující i převede automat ze stavů p, q buď v obou případech do koncového, nebo v obou případech do nekoncového stavu.

Lemma 1.5.6. *Nechť* $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$ *je konečný automat. Potom platí tato tvrzení:*

1. *Každá z relací* $\overset{i}{\sim}$, $i \geqq 0$, *je relací ekvivalence na množině stavů* Q. *Označme* $R_i = Q/\overset{i}{\sim}$ *rozklad* Q *podle* $\overset{i}{\sim}$.

2. *Pro každé* $i \geqq 0$ *je* R_{i+1} *zjemněním rozkladu* R_i, *tzn. pro libovolná* $p, q \in Q$ z $p \overset{i+1}{\sim} q$ *plyne* $p \overset{i}{\sim} q$.

3. *Jakmile pro nějaké* $i \geqq 0$ *je* $R_i = R_{i+1}$, *potom také* $R_i = R_{i+t}$ *pro libovolné* $t \geqq 1$.

4. *Jestliže* $|Q| = n$, *potom existuje* $k \leqq n - 1$ *takové, že* $R_k = R_{k+1}$.

5. *Pro každé k takové, že* $R_k = R_{k+1}$, *je relace* $\overset{k}{\sim}$ *totožná s relací* $\sim$, *tzn. pro libovolná* $p, q \in Q$ *je* $p \overset{k}{\sim} q \Leftrightarrow p \sim q$.

Důkaz. Tvrzení 1 a 2 plynou přímo z definice 1.5.5.

3. Z definice 1.5.5 a předpokladu $R_i = R_{i+1}$ plyne, že

$$p \overset{i+1}{\sim} q \Leftrightarrow p \overset{i}{\sim} q \text{ a zároveň pro všechna } a \in \Sigma \text{ je}$$

$$[\delta(p, a) \overset{i}{\sim} \delta(q, a)] \Leftrightarrow p \overset{i+1}{\sim} q \text{ a pro každé } a \in \Sigma \text{ je}$$

$$[\delta(p, a) \overset{i+1}{\sim} \delta(q, a)] \Leftrightarrow p \overset{i+2}{\sim} q .$$

Dokázali jsme tedy, že pro všechna $i \geqq 0$ z $R_i = R_{i+1}$ plyne $R_{i+1} = R_{i+2}$. Matematickou indukcí lze snadno dokázat, že z $R_i = R_{i+1}$ plyne $R_i = R_{i+t}$ pro libovolné $t \geqq 1$.

4. Počet tříd v rozkladu R_i označíme n_i. Jestliže $R_0, R_1, \ldots, R_k$ jsou navzájem různé, pak $1 \leqq n_0 < n_1 < \ldots < n_k \leqq n$. Z toho plyne, že $k \leqq n - 1$. Existuje tedy $k \leqq n - 1$ takové, že $n_k = n_{k+1}$, tj. $R_k = R_{k+1}$.

5. Jak jsme poznamenali, z definice 1.5.5 se snadno indukcí odvodí, že pro libovolné $i \geqq 0$ a libovolné stavy $p, q \in Q$ je $p \overset{i}{\sim} q$, právě když pro každé slovo $w \in \Sigma^*$ délky nejvýše i je $\delta(p, w) \in F \Leftrightarrow \delta(q, w) \in F$. Z toho už je vidět, že libovolné stavy p, q jsou ekvivalentní, právě když $p \overset{i}{\sim} q$ pro všechna i. Jestliže nyní $R_k = R_{k+1}$, potom na základě tvrzení 3 a 2 dostáváme, že pro libovolné stavy p, q je $p \overset{k}{\sim} q$, právě když $p \overset{i}{\sim} q$ pro všechna $i \geqq 0$. To znamená, že $p \overset{k}{\sim} q$, právě když $p \sim q$.

Algoritmus 1.5.7. Rozklad množiny stavů Q na $\sim$-třídy.

Nechť $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$ je konečný automat. Budeme užívat

značení z předchozího lemmatu a definice. Rozklad $Q/\sim$ se sestrojí těmito kroky:

1. Sestrojí se rozklad R_0; $i := 0$.

2. Sestrojí se R_{i+1} a provede se přiřazení $i := i + 1$.

3. Jestliže $R_i \neq R_{i-1}$, opakuje se operace 2, jinak se pokračuje operací 4.

4. Jako výsledný rozklad se označí R_i.

Příklad 1.5.8. Sestrojme rozklad $Q/\sim$ pro automat $\mathscr{A}$ zadaný tabulkou

	a	b
→ 1	2	3
2	2	4
← 3	3	5
4	2	7
← 5	6	3
← 6	6	6
7	7	4
8	2	3
9	9	4

Proberme podrobně konstrukci rozkladů R_i.

Rozklad R_0: množina stavů Q se rozpadá do dvou tříd. Jednu z nich tvoří koncové, druhou ostatní stavy. Označme R_0-třídy římskými číslicemi I a II a pro každou R_0-třídu sestavme tabulku, ze které vyplyne, do které R_0-třídy jsou stavy jednotlivými symboly převáděny.

Rozklad R_0 na třídy:

$$\mathrm{I} = \{1, 2, 4, 7, 8, 9\}, \quad \mathrm{II} = \{3, 5, 6\}.$$

Tabulky:

	a	b
1	I	II
2	I	I
4	I	I
7	I	I
8	I	II
9	I	I

	a	b
3	II	II
5	II	II
6	II	II

Rozklad R_1 bude zjemněním R_0. Podle definice 1.5.5 budou ve stejné třídě rozkladu R_1 ležet stavy, které jsou ze stejné R_0-třídy stejným vstupním symbolem převedeny vždy do stejné R_0-třídy.

Ve stejné třídě rozkladu R_1 budou ležet ty prvky, kterým odpovídají v tabulce stejné řádky.

Rozklad R_1 na třídy:

$$\text{I} = \{1, 8\}, \ \text{II} = \{2, 4, 7, 9\}, \ \text{III} = \{3, 5, 6\}.$$

Tabulky:

	a	b
1	II	III
8	II	III

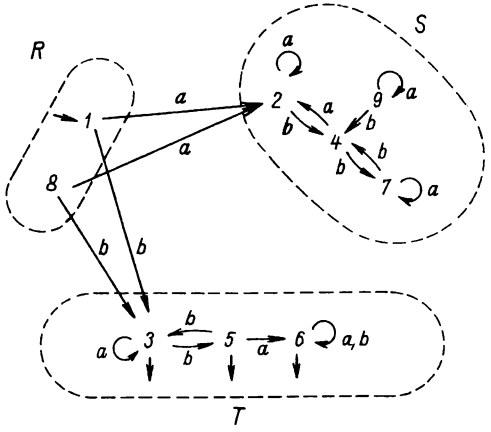
	a	b
2	II	II
4	II	II
7	II	II
9	II	II

	a	b
3	III	III
5	III	III
6	III	III

Rozklad R_2 na třídy:

$$\{1, 8\},\ \{2, 4, 7, 9\},\ \{3, 5, 6\}.$$

Tedy $R_2 = R_1$, a proto $Q/\sim = R_2$. (Viz též obr. 11.)



Obr. 11

Poznámka 1.5.9. Algoritmu 1.5.7 lze využít například i při hledání nejkratšího slova, které rozliší dva neekvivalentní stavy. Jestliže totiž pro nějaké stavy p, q je $p \overset{i}{\sim} q$ a není $p \overset{i+1}{\sim} q$, bude mít minimální rozlišující slovo délku $i + 1$. Navíc v tabulkách, jakých jsme používali v předchozím příkladu, lze okamžitě vyhledat nějaký symbol a_1 takový, že $\delta(q, a_1) = q_1$ a $\delta(p, a_1) = p_1$ pro nějaké stavy p_1, q_1 takové, že není $p_1 \overset{i}{\sim} q_1$, ale je $p_1 \overset{i-1}{\sim} q_1$. Podobně lze pro stavy p_1, q_1 vyčíst symbol a_2 takový, že $\delta(p_1, a_2) = p_2$, $\delta(q_1, a_2) = q_2$ a stavy p_2, q_2 jsou rozlišitelné právě slovem délky $i - 1$. Takto postupně získáme rozlišující slovo $a_1 a_2 \dots a_{i+1}$ pro stavy p a q.

Výsledku práce algoritmu 1.5.7 nad libovolným konečným automatem $\mathscr{A}$ je možné využít ke konstrukci automatu ekvivalentního s $\mathscr{A}$, jehož žádné dva stavy nejsou ekvivalentní.

Definice 1.5.10. Nechť $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$ je konečný automat a $\sim$ ekvivalence definovaná v 1.5.4. Definujme *podílový automat* $\mathscr{A}/\sim$ *automatu* $\mathscr{A}$ *podle ekvivalence* $\sim$ takto:

$$\mathscr{A}/\sim = (Q/\sim, \Sigma, \delta_\sim, [q_0], \{[q]; q \in F\}),$$

kde

$$\delta_{\sim}([q], a) = [\delta(q, a)]$$

pro každé $q \in Q$, $a \in \Sigma$.

($[q]$ označuje třídu rozkladu $Q/\sim$ obsahující q.)

Poznámka 1.5.11. Definice 1.5.10 je korektní, protože pro libovolné stavy p, $q \in Q$ takové, že $[p] = [q]$, je také $[\delta(q, a)] = [\delta(p, a)]$ pro každé $a \in \Sigma$, takže určení hodnoty $\delta([q], a)$ nezávisí na volbě reprezentanta q třídy $[q]$.

Příklad 1.5.12. Vraťme se k př. 1.5.8 a označme jednotlivé třídy rozkladu $Q/\sim$ takto:

$$R = \{1, 8\}, \; S = \{2, 4, 7, 9\}, \; T = \{3, 5, 6\}.$$

Potom automat $\mathscr{A}/\sim$ je dán tabulkou

	a	b
$\rightarrow R$	S	T
S	S	S
$\leftarrow T$	T	T

Lemma 1.5.13. *Nechť $\mathscr{A}$ je konečný automat a $\sim$ ekvivalence na jeho stavovém prostoru definovaná v* 1.5.4. *Potom podílový automat $\mathscr{A}/\sim$ je ekvivalentní automatu $\mathscr{A}$ a kromě toho žádné dva stavy automatu $\mathscr{A}/\sim$ nejsou vzájemně ekvivalentní. Jestliže navíc automat $\mathscr{A}$ neobsahuje nedosažitelné stavy, potom ani automat $\mathscr{A}/\sim$ neobsahuje nedosažitelné stavy.*

Důkaz. Budiž $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$ konečný automat, $w = a_1 \dots a_n$ libovolné slovo, $n \geqq 0$, a $q_0, \dots, q_n$ posloupnost stavů, do nichž se $\mathscr{A}$ postupně dostane při výpočtu nad slovem w, tzn. $q_i = \delta(q_0, a_1 \dots a_i)$. Potom se automat $\mathscr{A}/\sim$ během výpočtu nad w postupně dostává do stavů $[q_0], \dots, [q_n]$. Stav q_n je koncovým stavem automatu $\mathscr{A}$, právě když $[q_n]$ je koncovým stavem automatu $\mathscr{A}/\sim$. Proto $w \in L(\mathscr{A})$, právě když $w \in L(\mathscr{A}/\sim)$. Tedy skutečně $L(\mathscr{A}) = L(\mathscr{A}/\sim)$.

Dokažme druhé tvrzení lemmatu. Jestliže stav $[p]$ je ekvivalentní stavu $[q]$, znamená to, že

$$\delta_{\sim}([p], w) \in F_{\sim} \Leftrightarrow \delta_{\sim}([q], w) \in F_{\sim}$$

pro každé $w \in \Sigma^*$.

To podle definice automatu $\mathscr{A}/\sim$ znamená, že pro libovolné $w \in \Sigma^*$ je

$$[\delta(p, w)] \in F_{\sim} \Leftrightarrow [\delta(q, w)] \in F_{\sim} .$$

Stačí si uvědomit, že každý stav ekvivalentní s koncovým stavem musí být také koncový, abychom pro každé $w \in \Sigma^*$ měli

$$\delta(p, w) \in F \Leftrightarrow \delta(q, w) \in F ,$$

tzn. $p \sim q$.

Tím jsme dokázali, že z $[p] \sim [q]$ plyne, že $[p] = [q]$. Poslední tvrzení lemmatu je zřejmé, protože jakmile je v $\mathscr{A}$ dosažitelný stav q, potom v $\mathscr{A}/\sim$ je dosažitelný stav $[q]$.

Odstranění nedosažitelných stavů a přechod k podílovému automatu jsou dva kroky postačující k tzv. redukci automatu.

Definice 1.5.14. *Automat* $\mathscr{A}$ nazveme *redukovaným*, jestliže

1. nemá nedosažitelné stavy a
2. žádné dva jeho různé stavy nejsou ekvivalentní.

Definice 1.5.15. *Automat* $\mathscr{B}$ nazveme *reduktem automatu* $\mathscr{A}$, jestliže

1. $\mathscr{B}$ je ekvivalentní s $\mathscr{A}$ a
2. $\mathscr{B}$ je redukovaný.

Věta 1.5.16. *Ke každému konečnému automatu lze sestrojit nějaký jeho redukt.*

Důkaz. Jestliže z libovolného konečného automatu odstraníme nedosažitelné stavy a z takto vzniklého automatu $\mathscr{A}_1$ vytvoříme podílový automat $\mathscr{A}_1/\sim$, potom je podle 1.5.3 a 1.5.13 automat $\mathscr{A}/\sim$ reduktem $\mathscr{A}$.

Poznámka 1.5.17. Ponecháme na čtenáři, aby sám promyslel, že také přechod k podílovému automatu s následným odstra-

něním nedosažitelných stavů vede k reduktu výchozího automatu.

Ukážeme ještě, že redukt automatu je určen jednoznačně až na „pojmenování stavů".

Definice 1.5.18. Nechť

$$\mathscr{A}_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$$

a

$$\mathscr{A}_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$$

jsou dva konečné automaty. Pak zobrazení $h: Q_1 \to Q_2$ nazveme *automatovým homomorfismem*, právě když platí:

1. $h(q_1) = q_2$,
2. $h(\delta_1(q, a)) = \delta_2(h(q), a)$ pro každé $q \in Q_1$, $a \in \Sigma$,
3. $q \in F_1 \Leftrightarrow h(q) \in F_2$ pro každé $q \in Q_1$.

Jestliže h je navíc vzájemně jednoznačné zobrazení Q_1 na Q_2, potom se h nazývá *automatový izomorfismus*. Automaty $\mathscr{A}_1$ a $\mathscr{A}_2$ jsou pak izomorfní.

Věty 1.5.19. *Každé dva ekvivalentní redukované automaty jsou izomorfní.*

Důkaz. Nechť

$$\mathscr{A}_1 = (Q_1, \Sigma, \delta_1, q_1, F_1), \quad \mathscr{A}_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$$

jsou dva ekvivalentní redukované automaty. Sestrojíme vzájemně jednoznačné zobrazení $h: Q_1 \to Q_2$ a ukážeme, že je to automatový homomorfismus.

a) definice h: ukažme, že ke každému stavu $q \in Q_1$ existuje právě jeden stav $p \in Q_2$ takový, že

$$(*) \qquad \delta_1(q, w) \in F_1 \Leftrightarrow \delta_2(p, w) \in F_2 \quad \text{pro každé } w \in \Sigma^*.$$

Že takový stav p existuje nejvýše jeden, je zřejmé. Kdyby totiž existovaly dva různé takové stavy p_1, p_2, pak by z $(*)$ plynulo, že $p_1 \sim p_2$, což je spor s předpokladem, že $\mathscr{A}_2$ je redukovaný.

Zvolíme-li stav $q \in Q_1$ libovolně, je jisté, že existuje $u \in \Sigma^*$ tak, že $q = \delta_1(q_1, u)$, protože stav q je dosažitelný. Definujeme-li nyní $p = \delta_2(q_2, u)$, je (*) ekvivalentní tvrzení, že

$$\delta_1(q_1, uw) \in F_1 \Leftrightarrow \delta_2(q_2, uw) \in F_2$$

pro každé $w \in \Sigma^*$.

Toto tvrzení ovšem platí, protože $\mathscr{A}_1$, $\mathscr{A}_2$ jsou podle předpokladu ekvivalentní. Položíme-li tedy pro každé $q \in Q_1$ a $p \in Q_2$, $h(q) = p$, právě když q a p splňují (*), potom h je korektně definované zobrazení Q_1 do Q_2.

b) h je vzájemně jednoznačné zobrazení: stavy q a p mají v (*) symetrické postavení. Proto podle a) ke každému $p \in Q_2$ existuje právě jeden stav $q \in Q_1$ tak, že platí (*). Z tohoto důvodu je h vzájemně jednoznačné zobrazení.

c) h je homomorfismus:

1. Z (*) a z předpokladu, že $\mathscr{A}_1$ a $\mathscr{A}_2$ jsou ekvivalentní automaty, plyne, že $h(q_1) = q_2$.

2. Pro každé $q \in Q_1$ existuje $u \in \Sigma^*$ tak, že $\delta_1(q_1, u) = q$. Pro libovolné $a \in \Sigma$ pak platí $h(\delta_1(q, a)) = h(\delta_1(q_1, ua))$. Podle konstrukce h pak $h(\delta_1(q_1, ua)) = \delta_2(q_2, ua)$. Tedy

$$h(\delta_1(q, a)) = \delta_2(\delta_2(q_2, u), a).$$

Jelikož však

$$\delta_2(q_2, u) = h(\delta_1(q_1, u)) = h(q),$$

dostáváme, že

$$h(\delta_1(q, a)) = \delta_2(h(q), a).$$

3. Z (*) a definice h dostáváme speciálně pro $w = e$: $q \in F_1 \Leftrightarrow$ $\Leftrightarrow h(q) \in F_2$ pro libovolné $q \in Q_1$. Zobrazení h je tedy homomorfismus.

Důsledek 1.5.20. *Jestliže $\mathscr{A}$ je libovolný konečný automat a $\mathscr{A}_1$, $\mathscr{A}_2$ jeho libovolné redukty, potom $\mathscr{A}_1$ a $\mathscr{A}_2$ jsou izomorfní.*

Poznámka 1.5.21. Rozhodnutí odhlížet od konkrétní povahy stavů automatu, které jsme na začátku přijali, znamená, že z hlediska vlastností, kterými se zabýváme, nelze rozlišit dva izomorfní

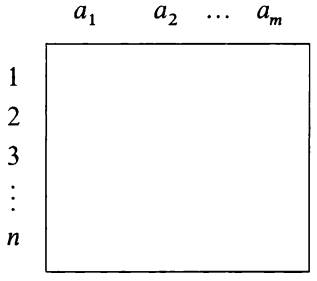
automaty. Proto je přirozené, že redukt automatu může být určen nejvýše až na izomorfismus, tzn. vlastně až na pojmenování stavů. Libovůli v pojmenování stavů lze vyloučit převedením automatů do tzv. normovaného tvaru.

Konečný automat $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$ bez nedosažitelných stavů se převede do *normovaného tvaru* procedurou, kterou nyní popíšeme. Všechny automaty s abecedou Σ se převádějí do normovaného tvaru v závislosti na určitém předem daném lineárním uspořádání množiny $\Sigma = \{a_1, \ldots, a_m\}$.

Výsledkem normalizační procedury je jistá tabulka automatu izomorfního s $\mathscr{A}$. Při konstrukci tabulky postupujeme takto:

Jestliže $|Q| = n$, bude množinou stavů nového automatu množina $\{1, 2, \ldots, n\}$.

Konstruovaná tabulka bude mít tento tvar:



K tomu, abychom ji vyplnili, stačí určit vzájemně jednoznačnou korespondenci h: $\{1, \ldots, n\} \to Q$. Přitom postupujeme tímto způsobem:

1. $h(1) =_{\mathrm{df}} q_0$.

2. Tabulku vyplňujeme po řádcích směrem shora dolů a každý řádek zleva doprava. Předpokládejme, že již máme vyplněn určitý počáteční úsek tabulky a máme stanoveno přiřazení $h(1), \ldots, h(i)$ pro nějaké $i \leqq n$. Pokračujeme ve vyplňování tabulky (podle přechodové funkce automatu $\mathscr{A}$) tak dlouho, dokud nenarazíme na nutnost zanést do tabulky stav odpovídající stavu q, který se ještě nevyskytuje mezi $h(1), \ldots, h(i)$. V tom případě položíme $h(i + 1) = q$ a na příslušné místo zaneseme $i + 1$. Po vyplnění

celé tabulky vyznačíme v levém sloupci počáteční stav, tj. 1, a koncové stavy.

Příklad 1.5.22. Normovaný tvar automatu daného tab. 1 je při uspořádání $a < b$ dán tab. 2 a při uspořádání $b < a$ tab. 3:

	a	b
A	B	A
$\leftrightarrow B$	D	C
C	A	D
$\leftarrow D$	A	B

Tab. 1

	a	b	
$\leftrightarrow 1$	2	3	$h(1) = B$
$\leftarrow 2$	4	1	$h(2) = D$
3	4	2	$h(3) = C$
4	1	4	$h(4) = A$

Tab. 2

	b	a	
$\leftrightarrow 1$	2	3	$h(1) = B$
2	3	4	$h(2) = C$
$\leftarrow 3$	1	4	$h(3) = D$
4	4	1	$h(4) = A$

Tab. 3

Za povšimnutí stojí, že při použití popsaného postupu na automat obsahující nedosažitelné stavy se tabulka jednoznačně zaplňuje až do okamžiku, kdy jsou vyčerpány všechny dosažitelné stavy. Zakončíme-li proces v tomto okamžiku, dostáváme normovaný tvar automatu vzniklého z výchozího automatu odstraněním nedosažitelných stavů. Lze tedy normalizaci automatu spojit s odstraněním nedosažitelných stavů. Tak také budeme postupovat v následujícím příkladu.

Příklad 1.5.23. Ověření ekvivalence automatů. Zjistíme, zda automaty $\mathscr{A}_1$, $\mathscr{A}_2$ zadané následujícími tabulkami jsou ekvivalentní.

$\mathscr{A}_1$:

	a	b
$\leftrightarrow q_0$	q_0	q_5
q_1	q_1	q_3
q_2	q_2	q_7
q_3	q_3	q_2
$\leftarrow q_4$	q_6	q_1
q_5	q_5	q_1
$\leftarrow q_6$	q_4	q_2
q_7	q_7	q_0

$\mathscr{A}_2$:

	b	a
A	G	H
B	A	B
C	D	E
D	B	D
E	D	C
F	E	F
$\leftrightarrow G$	F	G
H	G	A

Z věty 1.5.19 plyne, že $\mathscr{A}_1$, $\mathscr{A}_2$ jsou ekvivalentní, právě když se normované tvary jejich reduktů rovnají.

Jelikož budeme převádět redukt do normovaného tvaru, je výhodnější využít obráceného postupu zmíněného v poznámce 1.5.17 a nejprve provést sloučení ekvivalentních stavů.

Při konstrukci $\mathscr{A}_1/\sim$ algoritmem 1.5.7 postupně dostáváme tyto rozklady (ověřte):

R_0: $\{q_0, q_4, q_6\}, \{q_1, q_2, q_3, q_5, q_7\}$.

R_1: $\{q_0, q_4, q_6\}, \{q_1, q_2, q_3, q_5\}, \{q_7\}$.

R_2: $\{q_0, q_4, q_6\}, \{q_1, q_3, q_5\}, \{q_2\}, \{q_7\}$.

R_3: $\{q_0, q_4\}, \{q_6\}, \{q_1, q_5\}, \{q_3\}, \{q_2\}, \{q_7\}$.

R_4: $\{q_0\}, \{q_4\}, \{q_6\}, \{q_1\}, \{q_5\}, \{q_3\}, \{q_2\}, \{q_7\}$.

Žádné dva stavy automatu $\mathscr{A}_1$ tedy nejsou ekvivalentní. Zbývá už jen odstranit nedosažitelné stavy a vzniklý automat převést do normovaného tvaru. Oba úkoly lze provést zároveň, jak jsme už upozornili. Výsledkem je automat

	a	b	
$\leftrightarrow 1$	1	2	$h(1) = q_0$
2	2	3	$h(2) = q_5$
3	3	4	$h(3) = q_1$
4	4	5	$h(4) = q_3$
5	5	6	$h(5) = q_2$
6	6	1	$h(6) = q_7$

Zkonstruujme $\mathscr{A}_2/\sim$:

R_0: $\{A, B, C, D, E, F, H\}, \{G\}$.

R_1: $\{A, H\}, \{B, C, D, E, F\}, \{G\}$.

R_2: $\{A, H\}, \{B\}, \{C, D, E, F\}, \{G\}$.

R_3: $\{A, H\}, \{B\}, \{C, E, F\}, \{D\}, \{G\}$.

R_4: $\{A, H\}, \{B\}, \{C, E\}, \{F\}, \{D\}, \{G\}$.

R_5: $\{A, H\}, \{B\}, \{C, E\}, \{F\}, \{D\}, \{G\}$.

$R_4 = R_5$, to znamená, že R_5 je rovno rozkladu stavového prostoru automatu $\mathscr{A}_2$ podle ekvivalence $\sim$. Automat $\mathscr{A}_2/\sim$ popíšeme touto tabulkou, ve které je každá $\sim$-třída označena některým svým reprezentantem:

	b	a
A	G	A
B	A	B
C	D	C
F	C	F
D	B	D
$\leftrightarrow G$	F	G

Při převádění do normovaného tvaru je třeba dbát, aby symboly vstupní abecedy byly v tabulce stejně uspořádány, jako v prvním případě. Pak dostaneme tuto tabulku:

	a	b	
$\leftrightarrow 1$	1	2	$h(1) = G$
2	2	3	$h(2) = F$
3	3	4	$h(3) = C$
4	4	5	$h(4) = D$
5	5	6	$h(5) = B$
6	6	1	$h(6) = A$

V obou případech jsme dostali stejný normovaný tvar reduktu. Automaty $\mathscr{A}_1$ a $\mathscr{A}_2$ jsou tedy ekvivalentní.

Poznámka 1.5.24. Z věty 1.5.19 vyplývá, že *mezi všemi automaty ekvivalentními s daným automatem má každý jeho redukt nejmenší počet stavů.*

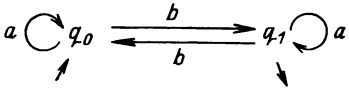
Důkaz. Během redukce automatu se nezvětšuje počet stavů. Kdyby tedy k nějakému automatu $\mathscr{A}_1$ existoval s ním ekvivalentní automat $\mathscr{A}_2$, který by měl menší počet stavů než redukt $\mathscr{A}^r$ automatu $\mathscr{A}_1$, měl by redukt $\mathscr{A}_2^r$ automatu $\mathscr{A}_2$ méně stavů než $\mathscr{A}^r$. To ale nemůže nastat, protože z ekvivalence $\mathscr{A}_1$ a $\mathscr{A}_2$ plyne, že $\mathscr{A}_1^r$ je izomorfní s $\mathscr{A}_2^r$, a tedy oba mají stejný počet stavů.

1.6. Realizace konečných automatů

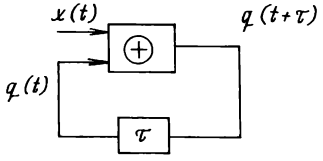
Jedno možné použití teorie automatů, na které ukazovaly naše dosavadní úvahy, spočívá v analýze skutečně existujících systémů. Při tomto typu úloh se nejprve získá automat jakožto abstraktní obraz studovaného systému a v další fázi se pomocí prostředků teorie automatů zkoumají vlastnosti tohoto systému.

Neméně důležitou je obrácená úloha – syntéza. Při ní se nejprve navrhne automat řešící určitou úlohu. Tato fáze končí u reprezentace automatu tabulkou, grafem apod. Reprezentace automatu pak slouží jako plán, podle něhož je sestrojován skutečný systém požadovaných vlastností. Začněme jednoduchým příkladem.

Příklad 1.6.1. Máme sestrojit zařízení schopné zjišťovat v posloupnostech symbolů a, b, zda obsahují lichý počet symbolů b. Je snadné navrhnout automat řešící tuto úlohu (obr. 12).



Obr. 12



Obr. 13

Budeme kódovat symbol a hodnotou 0 a symbol b hodnotou 1,

stav q_0 hodnotou 0

a

stav q_1 hodnotou 1.

Přechodová funkce automatu z obr. 12 je potom dána tabulkou, která odpovídá binární funkci $\oplus$ – sčítání mod 2. Automat lze realizovat podle schématu z obr. 13. Na tomto schématu je kromě bloku realizujícího funkci $\oplus$ ještě blok realizující zpoždění signálu o časovou jednotku τ. Na základě vstupního symbolu a stavu v čase t je vytvořena hodnota odpovídající následujícímu stavu. Tato hodnota je zpožděna o τ, aby se v čase $t + \tau$ spolu s dalším vstupním signálem dostala na vstup bloku realizujícího $\oplus$.

Tím je v hrubých rysech načrtnut problém syntézy. Proberme nyní tuto problematiku o něco podrobněji. Přejdeme od úlohy syntézy konečného automatu k úloze syntézy konečného sekvenčního stroje, což je úloha o něco obecnější.

Definice 1.6.2. *Konečným Mooreovým sekvenčním strojem* budeme rozumět každou pětici $\mathscr{M} = (Q, \Sigma, \Delta, \delta, \mu)$, kde Q, Σ, Δ jsou po řadě konečné neprázdné množiny stavů, vstupních a výstupních symbolů, δ je zobrazení $Q \times \Sigma \to Q$ (přechodová funkce) a μ je zobrazení $Q \to \Delta$ (*značkovací funkce*).

Jak je z definice vidět, disponuje Mooreův stroj výstupem, takže na posloupnost symbolů vstupní abecedy Σ postupně reaguje posloupností symbolů výstupní abecedy Δ. Takový výstup pochopitelně umožňuje suplovat roli koncových stavů. Místo množiny $F \subseteq Q$ lze totiž zavést značkovací funkci $\mu: Q \to \{0, 1\}$ předpisem

$$\mu(q) = \begin{cases} 1, & \text{jestliže } q \in F, \\ 0, & \text{jestliže } q \notin F. \end{cases}$$

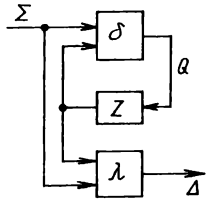
Jsme-li schopni řešit problém syntézy Mooreova stroje, je tím také řešen problém syntézy konečného automatu, neboť realizaci konečného automatu můžeme nahradit realizací odpovídajícího Mooreova stroje. Díky tomu, že výstupní abeceda může obsahovat více než dva symboly, je značkovací funkce obecně silnějším prostředkem než vymezení množiny koncových stavů.

Učiňme ještě jeden zobecňující krok.

Definice 1.6.3. *Konečným Mealyho sekvenčním strojem* rozumíme každou pětici $\mathscr{M} = (Q, \Sigma, \Delta, \delta, \lambda)$, kde Q, Σ, Δ jsou po řadě konečné neprázdné množiny stavů, vstupních a výstupních symbolů, δ je zobrazení $Q \times \Sigma \to Q$ (přechodová funkce) a λ je zobrazení $Q \times \Sigma \to \Delta$ (*výstupní funkce*).

Na rozdíl od Mooreova stroje je u Mealyho stroje výstup určen nejenom stavem, ale i vstupním symbolem. Opět je zřejmé, že Mealyho stroj je obecnějším prostředkem než stroj Mooreův, protože značkovací funkci $\mu: Q \to \Delta$ lze např. nahradit výstupní funkcí $\lambda: Q \times \Sigma \to \Delta$ definovanou předpisem $\lambda(q, x) = \mu(q)$ pro všechna $x \in \Sigma$, $q \in Q$.

Metodu realizace popíšeme pro nejobecnější z probraných automatů – Mealyho konečný sekvenční stroj, který v dalším výkladu budeme stručněji označovat jako sekvenční stroj nebo prostě jen stroj. Souvislosti jeho jednotlivých složek jsou zachyceny na obr. 14.



Obr. 14

Základním úkolem syntézy je realizace funkcí δ a λ. Zpožďovací elementy z, které zpožďují výstup funkce δ o časovou jednotku, budeme považovat za elementární, předem dané stavební prvky.

Zvláštní praktický význam mají sekvenční stroje, u nichž

$$Q = \{0,1\}^k,\ \Sigma = \{0,1\}^l,\ \Delta = \{0,1\}^m$$

pro nějaká k, l, m, protože funkce

$$\delta\colon \{0,1\}^{k+l} \to \{0,1\}^k$$

a

$$\lambda\colon \{0,1\}^{k+l} \to \{0,1\}^m$$

jsou booleovské funkce, které lze standardním způsobem vytvořit, např. pomocí kombinačních sítí. Spojením zpožďovacích elementů s kombinačními sítěmi pro δ a λ podle obr. 14 vzniká sekvenční obvod realizující příslušný abstraktní stroj.

Příklad 1.6.4. Ukažme si na příkladu základní fáze realizace sekvenčního stroje sekvenčním obvodem. Výchozí sekvenční stroj s $Q = \{A, B, C\}$, $\Sigma = \{1, 2, 3\}$, $\Delta = \{0, 1\}$ je dán těmito tabulkami:

δ:	1	2	3
A	A	B	B
B	A	B	C
C	B	B	C

λ:	1	2	3
A	0	0	0
B	0	1	1
C	0	1	1

Prvním krokem je zakódování symbolů a stavů binárními vektory. Výstupním symbolům ponechme v tomto případě jejich binární hodnoty a pro ostatní zvolme kódování

$$A \to (0,0), \qquad 1 \to (0,0),$$

$$B \to (1,0), \qquad 2 \to (0,1),$$

$$C \to (1,1), \qquad 3 \to (1,1).$$

Obecně může být volba kódování určena dodatečnými vnějšími požadavky na navrhovaný systém, nebo může být ponechána na libovůli návrháře.

K zakódování n-prvkové množiny je třeba volit vektory délky $[\log_2 n]$. Jestliže n není mocninou dvojky, budou některé vektory nevyužity. V našem případě dostáváme tyto tabulky:

$\hat{\delta}$:	(0, 0)	(0, 1)	(1, 0)	(1, 1)
(0, 0)	(0, 0)	(1, 0)	–	(1, 0)
(0, 1)	–	–	–	–
(1, 0)	(0, 0)	(1, 0)	–	(1, 1)
(1, 1)	(1, 0)	(1, 0)	–	(1, 1)

$\hat{\lambda}$:	(0, 0)	(0, 1)	(1, 0)	(1, 1)
(0, 0)	0	0	–	0
(0, 1)	–	–	–	–
(1, 0)	0	1	–	1
(1, 1)	0	1	–	1

Prázdná místa v tabulkách označují argumenty, pro něž hodnota příslušné funkce není výchozím zadáním určena. Proto je možné vyplnit je libovolně. Jedno z možných vyplnění je toto:

$\hat{\delta}$:	(0, 0)	(0, 1)	(1, 0)	(1, 1)
(0, 0)	(0, 0)	(1, 0)	(0, 0)	(1, 0)
(0, 1)	(1, 0)	(1, 0)	(1, 0)	(1, 0)
(1, 0)	(0, 0)	(1, 0)	(0, 1)	(1, 1)
(1, 1)	(1, 0)	(1, 0)	(1, 1)	(1, 1)

$\hat{\lambda}$:	(0, 0)	(0, 1)	(1, 0)	(1, 1)
(0, 0)	0	0	0	0
(0, 1)	0	0	0	0
(1, 0)	0	1	0	1
(1, 1)	0	1	0	1

Tím jsme získali dvě booleovské funkce

$$\hat{\delta}\colon \{0,1\}^4 \to \{0,1\}^2$$

a

$$\hat{\lambda}\colon \{0,1\}^4 \to \{0,1\},$$

které můžeme přepsat do tabulky takto:

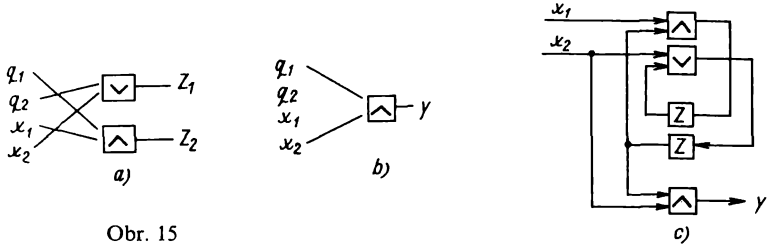
				$\hat{\delta}$		$\hat{\lambda}$
q_1	q_2	x_1	x_2	z_1	z_2	y
0	0	0	0	0	0	0
0	0	0	1	1	0	0
0	0	1	0	0	0	0
0	0	1	1	1	0	0
0	1	0	0	1	0	0
0	1	0	1	1	0	0
0	1	1	0	1	0	0
0	1	1	1	1	0	0
1	0	0	0	0	0	0

				$\hat{\delta}$		$\hat{\lambda}$
q_1	q_2	x_1	x_2	z_1	z_2	y
1	0	0	1	1	0	1
1	0	1	0	0	1	0
1	0	1	1	1	1	1
1	1	0	0	1	0	0
1	1	0	1	1	0	1
1	1	1	0	1	1	0
1	1	1	1	1	1	1

Tab. 4

Metody umožňující konstrukci co možná nejmenší kombinační sítě pro realizaci dané booleovské funkce jsou podrobně rozpracovány, ale přesahují rámec této knihy. Proto se spokojme s konstatováním, že obě funkce jsou realizovány sítěmi na obr. 15a) a b), jak se čtenář může snadno přesvědčit.

Schéma výsledného sekvenčního obvodu je na obr. 15c).



Obr. 15

Změnou kódování i změnou v doplňování neurčených hodnot pochopitelně dostáváme odlišnou realizaci, která se může podstatně lišit složitostí kombinačních sítí pro přechodovou i výstupní funkci.

Zrekapitulujme si na závěr postup realizace. Stavy a abecedy výchozího stroje

$$\mathcal{M}_1 = (Q_1, \Sigma_1, \Delta_1, \delta_1, \lambda_1)$$

jsme zakódovali ve stavech a abecedách jistého stroje

$$\mathcal{M}_2 = (Q_2, \Sigma_2, \Delta_2, \delta_2, \lambda_2),$$

který odrážel strukturu stroje $\mathcal{M}_2$. Proto pojem realizace stroje těsně souvisí s pojmem homomorfismu sekvenčních strojů, který budeme definovat.

Definice 1.6.5. Nechť

$$\mathcal{M}_1 = (Q_1, \Sigma_1, \Delta_1, \delta_1, \lambda_1)$$

a

$$\mathcal{M}_2 = (Q_2, \Sigma_2, \Delta_2, \delta_2, \lambda_2)$$

jsou dva sekvenční stroje a

$$h = (h_Q, h_\Sigma, h_\Delta)$$

je trojice zobrazení

$$h_Q: Q_1 \to Q_2, \quad h_\Sigma: \Sigma_1 \to \Sigma_2, \quad h_\Delta: \Delta_1 \to \Delta_2.$$

Jestliže

1. $h_Q(\delta_1(q, a)) = \delta_2(h_Q(q), h_\Sigma(a))$

a

2. $h_\Delta(\lambda_1(q, a)) = \lambda_2(h_Q(q), h_\Sigma(a))$

pro každé $q \in Q_1$, $a \in \Sigma_1$, potom se h nazývá *homomorfismus stroje* $\mathcal{M}_1$ *do* $\mathcal{M}_2$.

Jestliže všechna tři zobrazení h_Q, h_Σ, h_Δ jsou prostá, potom h je *prostý homomorfismus* $\mathcal{M}_1$ *do* $\mathcal{M}_2$.

Definice 1.6.6. Nechť $\mathcal{M}_1$, $\mathcal{M}_2$ jsou sekvenční stroje. Řekněme, že $\mathcal{M}_2$ *realizuje* $\mathcal{M}_1$, jestliže existuje prostý homomorfismus $\mathcal{M}_1$ do $\mathcal{M}_2$.

Naše definice odhlíží od mnoha problémů spjatých např. s konkrétní technickou realizací automatů jakožto elektronických zařízení a vystihuje jen ty vlastnosti realizace, které jsou na úrovni abstrakce, kterou jsme zvolili na začátku této kapitoly. Takovéto zjednodušení umožňuje se orientovat i ve značně složitých problémech, jakým je i dekompozice sekvenčních strojů, kterou se budeme zabývat v následujícím článku.

Možnost nahradit rozsáhlý sekvenční stroj systémem několika menších, vzájemně propojených strojů, s sebou přináší některé výhody. Tak například v případě výskytu poruchy stačí lokalizovat poruchu v jedné složce a vyměnit pouze tuto část.

Zde se omezíme jen na dva jednoduché typy spojování strojů – sériové spojení a paralelní spojení.

Definice 1.7.1. Nechť

$$\mathcal{M}_1 = (Q_1, \Sigma_1, \Delta_1, \delta_1, \lambda_1)$$

a

$$\mathcal{M}_2 = (Q_2, \Sigma_2, \Delta_2, \delta_2, \lambda_2)$$

jsou sekvenční stroje a $\Delta_1 \subseteq \Sigma_2$. Potom *sériové spojení strojů* $\mathcal{M}_1, \mathcal{M}_2$ je stroj

$$\mathcal{M}_1 \ominus \mathcal{M}_2 = (Q_1 \times Q_2, \Sigma_1, \Delta_2, \delta, \lambda),$$

kde

$$\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, \lambda_1(q_1, a)))$$

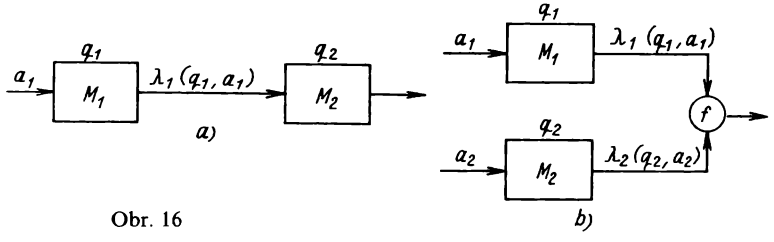
a

$$\lambda((q_1, q_2), a) = \lambda_2(q_2, \lambda_1(q_1, a))$$

pro každé

$$q_1 \in Q_1\,,\ q_2 \in Q_2\,,\ a \in \Sigma\,.$$

Sériové spojení strojů můžeme schematicky znázornit na obr. 16a). Považujeme-li systém znázorněný na tomto obrázku za jediný sekvenční stroj, potom stav tohoto systému je určen dvěma složkami – stavem stroje $\mathcal{M}_1$ a stavem stroje $\mathcal{M}_2$. Proto je stavovým prostorem nového stroje množina $Q_1 \times Q_2$ a stejně



Obr. 16

přirozeně se stanoví funkce δ a λ tak, jak je tomu v definici 1.7.1. Analogicky dojdeme k definici paralelního spojení strojů.

Definice 1.7.2. Nechť

$$\mathcal{M}_1 = (Q_1, \Sigma_1, \Delta_1, \delta_1, \lambda_1)$$

a

$$\mathcal{M}_2 = (Q_2, \Sigma_2, \Delta_2, \delta_2, \lambda_2)$$

jsou sekvenční stroje a

$$f\colon \Delta_1 \times \Delta_2 \to \Delta$$

pro nějakou abecedu Δ. Potom *paralelní spojení strojů* $\mathcal{M}_1$ a $\mathcal{M}_2$ *se spojkou* f je stroj

$$\mathcal{M}_1 \mid f \mid \mathcal{M}_2 = (Q_1 \times Q_2, \Sigma_1 \times \Sigma_2, \Delta, \delta, \lambda),$$

kde

$$\delta((q_1, q_2), (a_1, a_2)) = (\delta_1(q_1, a_1), \delta_2(q_2, a_2))$$

a

$$\lambda((q_1, q_2), (a_1, a_2)) = f(\lambda_1(q_1, a_1), \lambda_2(q_2, a_2))$$

pro každé

$$q_1 \in Q_1\,,\ q_2 \in Q_2\,,\ a_1 \in \Sigma_1\,,\ a_2 \in \Sigma_2\,.$$

Paralelní spojení můžeme schematicky znázornit na obr. 16b).

V dalším výkladu ukážeme, jak určit, zda daný stroj má netriviální sériovou či paralelní dekompozici.

Definice 1.7.3. Nechť $\mathcal{M}_1$, $\mathcal{M}_2$ a $\mathcal{M}$ jsou sekvenční stroje. Jestliže sériové spojení $\mathcal{M}_1 \ominus \mathcal{M}_2$ realizuje stroj $\mathcal{M}$, řekneme, že $\mathcal{M}_1 \ominus \mathcal{M}_2$ je *sériovou dekompozicí stroje* $\mathcal{M}$.

Definice 1.7.4. Nechť $\mathcal{M}_1$, $\mathcal{M}_2$ a $\mathcal{M}$ jsou sekvenční stroje. Jestliže paralelní spojení se spojkou f $\mathcal{M}_1 \mid f \mid \mathcal{M}_2$ realizuje stroj $\mathcal{M}$, potom řekneme, že $\mathcal{M}_1 \mid f \mid \mathcal{M}_2$ je *paralelní dekompozicí stroje* $\mathcal{M}$.

Sériová, resp. paralelní dekompozice se nazývá *netriviální*, jestliže každý ze strojů $\mathcal{M}_1$, $\mathcal{M}_2$ má méně stavů než $\mathcal{M}$.

Otázku existence netriviální sériové či paralelní dekompozice převedeme na otázku, zda existují určité rozklady stavového prostoru stroje.

Dohodněme se nejprve na značení. Jestliže π je rozklad nějaké množiny Q, potom zápis $q \equiv q'(\pi)$ bude značit, že prvky q, q' leží ve stejné třídě rozkladu π.

Definujme operace průseku . a spojení + rozkladů, které z libovolných rozkladů π_1, π_2 množiny tvoří rozklady $\pi_1 . \pi_2$ a $\pi_1 + \pi_2$ téže množiny. Tyto operace jsou definovány takto:

$$q \equiv q'(\pi_1 . \pi_2) \Leftrightarrow_{\mathrm{df}} q \equiv q'(\pi_1) \,\&\, q \equiv q'(\pi_2),$$
$$q \equiv q'(\pi_1 + \pi_2) \Leftrightarrow_{\mathrm{df}}$$

existují $q_1, \ldots, q_n \in Q$ taková, že

$$q_1 = q \,\&\, q_n = q' \,\&$$

pro všechna i, $1 \leqq i \leqq n$, je

$$q_i \equiv q_{i+1}(\pi_1) \text{ nebo } q_i \equiv q_{i+1}(\pi_2).$$

Symbolem O budeme označovat rozklad na jednobodové množiny a symbolem I rozklad tvořený jedinou třídou zahrnující celou množinu Q. Tyto dva rozklady nazýváme *triviálními*.

Každý rozklad stavového prostoru sekvenčního stroje můžeme intuitivně spojovat s určitým stupněm přesnosti lokalizace stavů (stav je lokalizován pouze třídou ekvivalence). Zvláštní důležitost mají rozklady, zachovávající tento stupeň přesnosti během libovolného výpočtu stroje, které nyní budeme definovat.

Definice 1.7.5. Řekneme, že rozklad π stavového prostoru sekvenčního stroje

$$\mathcal{M} = (Q, \Sigma, \Delta, \delta, \lambda)$$

má *substituční vlastnost* (je to sv-*rozklad*), jestliže pro každé q, $q' \in Q$,

$$q \equiv q'(\pi) \Rightarrow \delta(q, a) \equiv \delta(q', a)(\pi)$$

pro každé $a \in \Sigma$.

V systému dvou paralelně spojených strojů [viz obr. 16b)] pracují obě složky nezávisle na sobě. Stanovíme-li tedy částečně

stav systému tak, že určíme stav pouze jedné složky, bude tento stupeň přesnosti zachován během libovolného výpočtu. Navíc stav obou složek jednoznačně určuje stav celého systému. Tato dvě pozorování jsou základem následujícího tvrzení.

Věta 1.7.6. *Stroj $\mathscr{M}$ má netriviální paralelní dekompozici, právě když existují dva netriviální* sv-*rozklady π, τ stavového prostoru stroje $\mathscr{M}$ takové, že $\pi . \tau = 0$.*

Důkaz. Nechť $\mathscr{M} = (Q, \Sigma, \Delta, \delta, \lambda)$ je Mealyho stroj.

1. Předpokládejme, že existují stroje

$$\mathscr{M}_1 = (Q_1, \Sigma_1, \Delta_1, \delta_1, \lambda_1)$$

a

$$\mathscr{M}_2 = (Q_2, \Sigma_2, \Delta_2, \delta_2, \lambda_2)$$

a f tak, že

$$\max(|Q_1|, |Q_2|) < |Q|,$$

a existuje prostý homomorfismus

$$g = (g_Q, g_\Sigma, g_\Delta)$$

stroje $\mathscr{M}$ do $\mathscr{M}_1 \mid f \mid \mathscr{M}_2$.

Definujme na Q rozklady π a τ takto:

$q \equiv q'(\pi) \Leftrightarrow_{\text{df}}$ první složka $g_Q(q)$ se rovná první složce $g_Q(q')$,

$q \equiv q'(\tau) \Leftrightarrow_{\text{df}}$ druhá složka $g_Q(q)$ se rovná druhé složce $g_Q(q')$.

Rozklady π a τ jsou netriviální, protože stroje $\mathscr{M}_1$ a $\mathscr{M}_2$ mají méně stavů než $\mathscr{M}$.

Přímo z definice π a τ a ze skutečnosti, že zobrazení g_Q je prosté, plyne $\pi . \tau = 0$.

Oba rozklady mají zřejmě substituční vlastnost.

2. Nechť na stavovém prostoru stroje $\mathscr{M}$ existují netriviální sv-rozklady π a τ takové, že $\pi . \tau = 0$. Definujme stroje

$$\mathscr{M}_1 = (\pi, \Sigma, \pi \times \Sigma, \delta_\pi, \text{id})$$

a

$$\mathscr{M}_2 = (\tau, \Sigma, \tau \times \Sigma, \delta_\tau, \text{id})$$

předpisem

$$\delta_\pi([q]_\pi, a) = [\delta(q, a)]_\pi,$$
$$\delta_\tau([q]_\tau, a) = [\delta(q, a)]_\tau$$

pro každé $q \in Q$ a $a \in \Sigma$.

Oba stroje jsou definovány korektně, protože π a τ jsou sv-rozklady. Dále definujme zobrazení

$$f: (\pi \times \Sigma) \times (\tau \times \Sigma) \to \Delta$$

předpisem

$$f(([q]_\pi, a), ([q]_\tau, a)) = \lambda(q, a)$$

pro každé $q \in Q$, $a \in \Sigma$. Korektnost definice plyne z předpokladu $\pi \cdot \tau = 0$. V ostatních bodech dodefinujme funkci f libovolně.

Potom stroj $\mathscr{M}$ lze vnořit do $\mathscr{M}_1 \mid f \mid \mathscr{M}_2$ prostým homomorfismem

$$g = (g_Q, g_\Sigma, g_\Delta),$$

kde

$$g_Q(q) = ([q]_\pi, [q]_\tau)$$

pro libovolné $q \in Q$, $g_\Sigma(a) = (a, a)$

pro každé $a \in \Sigma$ a $g_\Delta(b) = b$

pro každé $b \in \Delta$.

Analogicky lze charakterizovat možnost sériové dekompozice.

Věta 1.7.7. *Sekvenční stroj má netriviální sériovou dekompozici, právě když existuje netriviální* sv-*rozklad jeho množiny stavů.*

Důkaz. 1. Nechť

$$\mathscr{M} = (Q, \Sigma, \Delta, \delta, \lambda)$$

má netriviální sériovou dekompozici

$$\mathscr{M}_1 \ominus \mathscr{M}_2,$$

kde

$$\mathscr{M}_1 = (Q_1, \Sigma_1, \Delta_1, \delta_1, \lambda_1),$$
$$\mathscr{M}_2 = (Q_2, \Sigma_2, \Delta_2, \delta_2, \lambda_2).$$

Nechť

$$h = (h_Q, h_\Sigma, h_\Delta)$$

je prostý homomorfismus $\mathscr{M}$ do $\mathscr{M}_1 \ominus \mathscr{M}_2$. Definujme rozklad π množiny Q tímto předpisem:

$$q \equiv q'(\pi) \Leftrightarrow_{\text{df}} q_1 = q'_1 ,$$

kde

$$h_Q(q) = (q_1, q_2) \text{ a } h_Q(q') = (q'_1, q'_2) .$$

Snadno se ověří, že π má substituční vlastnost. π není triviální, neboť

$$h_Q : Q \to Q_1 \times Q_2$$

je prosté zobrazení a

$$\max(|Q_1|, |Q_2|) < |Q| .$$

2. Nechť π je netriviální sv-rozklad množiny Q. Sestrojme rozklad τ množiny Q tak, že $\pi . \tau = 0$ a τ má tolik tříd, kolik má největší třída z π prvků. Takový rozklad (nepožadujeme, aby měl substituční vlastnost) utvoříme například tak, že z každé třídy rozkladu π vybereme po jednom prvku. Takto získané prvky budou tvořit jednu třídu rozkladu τ. Poté ze zbývajících neprázdných tříd π vybereme opět po jednom prvku a z nich utvoříme další třídu τ. Postup opakujeme až do vyčerpání všech tříd π.

Nyní položíme

$$\mathscr{M}_1 = (\pi, \Sigma, \pi \times \Sigma, \delta_\pi, \text{id}),$$

$$\mathscr{M}_2 = (\tau, \pi \times \Sigma, \Delta, \delta_2, \lambda_2),$$

kde δ_π definujeme takto:

$$\delta_\pi([q]_\pi, a) =_{\text{df}} [\delta(q, a)]_\pi$$

pro každé $q \in Q$, $a \in \Sigma$.

Zobrazení δ_π je definováno korektně, protože π má substituční vlastnost. δ_2 nelze takto definovat, protože τ nemusí mít substi-

tuční vlastnost. Tato přechodová funkce musí proto používat informace poskytované na výstupu stroje $\mathscr{M}_1$. Definujeme

$$\delta_2(T_i, (P, a)) = T_j$$

pro všechna

$$T_i,\ T_j \in \tau \text{ a } P \in \pi$$

taková, že

$$\delta(T_i \cap P, a) \in T_j .$$

V ostatních bodech lze δ_2 dodefinovat libovolně, protože tyto hodnoty nebudou při sériovém spojení s $\mathscr{M}_1$ využity.

$$\lambda_2(T, (P, a)) =_{\text{df}} \lambda(T \cap P, a)$$

pro všechna

$$T \in \tau \text{ a } P \in \pi$$

taková, že

$$T \cap P \neq \emptyset .$$

V ostatních bodech dodefinujeme λ_2 libovolně.

Funkce δ_2 i λ_2 jsou definovány korektně, protože $T \cap P$ je pro libovolná $T \in \tau$, $P \in \pi$ nejvýše jednobodová množina. $\mathscr{M}_1$ i $\mathscr{M}_2$ mají méně stavů než $\mathscr{M}$. Konečně, $\mathscr{M}$ lze do $\mathscr{M}_1 \ominus \mathscr{M}_2$ vnořit prostým homomorfismem

$$h = (h_Q, h_\Sigma, h_\Delta),$$

kde

$$h_\Sigma = \mathrm{id}_\Sigma,\ h_\Delta = \mathrm{id}_\Delta \text{ a } h_Q(q) = (P, T),$$

kde $P \in \pi$ a $T \in \tau$ jsou takové, že $P \cap T = \{q\}$.

Ověření, že se jedná o prostý homomorfismus, přenecháme čtenáři jako cvičení.

Předcházející věty ukazují, že je možné získat všechny možné sériové a paralelní dekompozice stroje, jakmile známe všechny sv-rozklady jeho stavového prostoru. Toto zjištění získá na ceně,

ukážeme-li dostatečně jednoduchý způsob konstrukce sv-rozkladů.

Metoda, se kterou se seznámíme, se opírá o následující větu.

Věta 1.7.8. *Jestliže* π *a* τ *jsou* sv-*rozklady stavového prostoru sekvenčního stroje, potom* $\pi . \tau$ *a* $\pi + \tau$ *jsou také* sv-*rozklady. Mezi* sv-*rozklady patří i triviální rozklady* I *a* O.

Důkaz. Nechť π a τ jsou sv-rozklady stavového prostoru sekvenčního stroje $\mathscr{M} = (Q, \Sigma, \Delta, \delta, \lambda)$.

1. Nechť $q \equiv q'(\pi . \tau)$. Potom $q \equiv q'(\pi)$ a $q \equiv q'(\tau)$. Tedy pro každé $a \in \Sigma$ je

$$\delta(q, a) \equiv \delta(q', a)(\pi)$$

a zároveň

$$\delta(q, a) \equiv \delta(q', a)(\tau).$$

Odtud plyne, že pro každé $a \in \Sigma$ je

$$\delta(q, a) \equiv \delta(q', a)(\pi . \tau).$$

To znamená, že $\pi . \tau$ je sv-rozklad.

2. Nechť $q \equiv q'(\pi + \tau)$, tj. existují stavy $q_1, \ldots, q_n \in Q$ takové, že

$$q_1 = q\,,\ q_n = q' \ \&\ q_i \equiv q_{i+1}(\pi)$$

nebo

$$q_i \equiv q_{i+1}(\tau)$$

pro všechna i, $1 \leqq i \leqq n$.

Pro tato $q_1, \ldots, q_n$ platí, že pro libovolné i, $1 \leqq i \leqq n$, a libovolné $a \in \Sigma$ je

$$\delta(q_i, a) \equiv \delta(q_{i+1}, a)(\pi)$$

nebo

$$\delta(q_i, a) \equiv \delta(q_{i+1}, a)(\tau).$$

Pro každé $a \in \Sigma$ je tedy

$$\delta(q, a) \equiv \delta(q'a)(\pi + \tau).$$

Odtud plyne, že $\pi + \tau$ je sv-rozklad.

3. I a O jsou sv-rozklady, jak plyne přímo z definice.

Algoritmus 1.7.9. Konstrukce sv-rozkladů. Pro daný sekvenční stroj označme symbolem $\langle q_1, q_2 \rangle$ nejmenší (tj. nejjemnější) sv-rozklad π takový, že $q_1 \equiv q_2(\pi)$. Všechny sv-rozklady stavového prostoru Q získáme takto:

1. pro každou dvojici $q_1, q_2 \in Q$ sestrojíme

$$\langle q_1, q_2 \rangle,$$

2. utvoříme všechna možná konečná spojení rozkladů typu

$$\langle q_1, q_2 \rangle.$$

Z věty 1.7.8 plyne, že uvedeným způsobem po konečném počtu kroků získáme všechny sv-rozklady. Postup budeme ilustrovat jednoduchým příkladem.

Příklad 1.7.10. Popišme všechny sv-rozklady stavového prostoru stroje s přechodovou funkcí danou touto tabulkou:

	a	b	c	d	f
1	2	1	5	8	3
2	1	2	6	7	4
3	4	3	6	6	1
4	3	4	5	5	2
5	5	6	3	4	7
6	6	5	4	3	8
7	7	8	4	2	5
8	8	7	3	1	6

Sestrojme nejprve všechny rozklady $\langle i, j \rangle$, $1 \leqq i < j \leqq 8$.

Rozklad $\langle 1, 2 \rangle$ získáme takto: jestliže $1 \equiv 2(\pi)$ pro nějaký sv-rozklad π, potom $3 \equiv 4(\pi)$, $5 \equiv 6(\pi)$ a $7 \equiv 8(\pi)$, neboť $3 = \delta(1, f)$ a $4 = \delta(2, f)$, dále $5 = \delta(1, c)$ a $6 = \delta(2, c)$ a konečně $7 = \delta(2, d)$ a $8 = \delta(1, d)$. Rozklad $\{1, 2\}, \{3, 4\}, \{5, 6\}, \{7, 8\}$ už je sv-rozkladem, jak se může čtenář přesvědčit. Je tedy nejmenším sv-rozkladem obsahujícím 1 a 2 ve stejné třídě.

Podobně při vyšetřování $\langle 1, 3\rangle$ $1 \equiv 3$ implikuje $2 \equiv 4$, $5 \equiv 6$, $6 \equiv 8$. Ekvivalence $2 \equiv 4$ navíc implikuje $5 \equiv 7$. Máme již $5 \equiv 6 \equiv 7 \equiv 8$. Z toho plyne, že $1 \equiv 2 \equiv 3 \equiv 4$. Rozklad

$$\{1, 2, 3, 4\}, \{5, 6, 7, 8\}$$

už je sv-rozklad. Takto vytvoříme všechny tyto základní sv-rozklady:

$$\langle 1, 2\rangle = \{1, 2\}, \{3, 4\}, \{5, 6\}, \{7, 8\},$$

$$\langle 1, 3\rangle = \{1, 2, 3, 4\}, \{5, 6, 7, 8\},$$

$$\langle 1, 4\rangle = \{1, 4\}, \{2, 3\}, \{5, 8\}, \{6, 7\},$$

$$\langle 1, 5\rangle = \langle 1, 6\rangle = I = \{1, 2, 3, 4, 5, 6, 7, 8\},$$

$$\langle 1, 7\rangle = \langle 1, 8\rangle = \{1, 2, 7, 8\}, \{3, 4, 5, 6\},$$

$$\langle 2, 3\rangle = \langle 1, 4\rangle,$$

$$\langle 2, 4\rangle = \langle 1, 3\rangle,$$

$$\langle 2, 5\rangle = \langle 2, 6\rangle = I,$$

$$\langle 2, 7\rangle = \langle 2, 8\rangle = \langle 1, 7\rangle,$$

$$\langle 3, 4\rangle = \langle 1, 2\rangle,$$

$$\langle 3, 5\rangle = \langle 3, 6\rangle = \langle 1, 7\rangle,$$

$$\langle 3, 7\rangle = \langle 3, 8\rangle = I,$$

$$\langle 4, 5\rangle = \langle 4, 6\rangle = \langle 1, 7\rangle,$$

$$\langle 4, 7\rangle = \langle 4, 8\rangle = I,$$

$$\langle 5, 6\rangle = \langle 1, 2\rangle,$$

$$\langle 5, 7\rangle = \langle 1, 3\rangle,$$

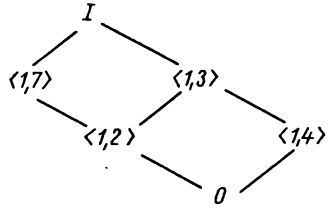
$$\langle 5, 8\rangle = \langle 1, 4\rangle,$$

$$\langle 6, 7\rangle = \langle 1, 4\rangle,$$

$$\langle 6, 8\rangle = \langle 1, 3\rangle,$$

$$\langle 7, 8\rangle = \langle 1, 2\rangle.$$

Získali jsme 6 základních sv-rozkladů: O, I, $\langle 1,2\rangle$, $\langle 1,3\rangle$, $\langle 1,4\rangle$, $\langle 1,7\rangle$. Spojením těchto rozkladů žádné nové sv-rozklady nezískáme. Strukturu sv-rozkladů na množině Q je možné znázornit diagramem na obr. 17.



Obr. 17

2. METODY NÁVRHU KONEČNÝCH AUTOMATŮ

Teoretické nástroje, se kterými se seznámíme v této kapitole, lze často s výhodou využít při návrhu konečných automatů. Jejich použití zpravidla umožňuje navrhnout automat s menší mentální námahou a usnadňuje kontrolu správnosti návrhu.

ZÁKLADNÍ ČÁST

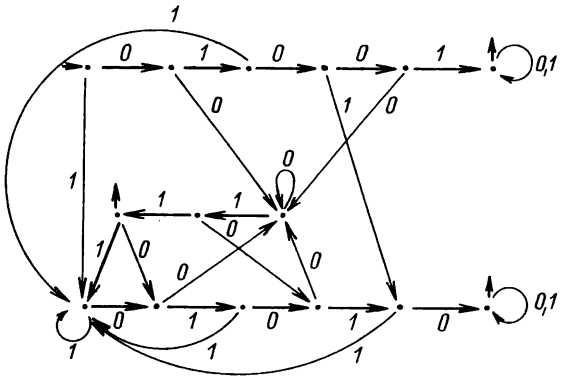
2.1. Nedeterministické konečné automaty

Typickou situaci, ve které při návrhu konečného automatu pomáhá pojem nedeterministického automatu, probereme na jednoduchém příkladu.

Příklad 2.1.1. Navrhněme konečný automat přijímající právě všechna slova v abecedě $\{0, 1\}$, která splňují alespoň jednu z těchto podmínek:

1. slovo začíná úsekem (prefixem) 01001,
2. slovo obsahuje 01010 jako podslovo (tzn. je tvaru $u01010v$, kde $u, v \in \{0, 1\}^*$),
3. slovo končí úsekem (sufixem) 0011.

Automat rozpoznávající tento jazyk je reprezentován na obr. 18.



Obr. 18

Přesto, že se jedná o jednoduchý příklad, vyžaduje návrh příslušného automatu značnou dávku pozornosti. Ani ověření, že automat skutečně rozpoznává požadovaný jazyk, není úplně jednoduchou záležitostí, jak se sami můžete přesvědčit. U komplikovanějších případů, zahrnujících např. větší počet podmínek nebo delší kontrolovaná podslova, by se návrh podobného konečného automatu mohl stát skutečným problémem a existovalo by reálné nebezpečí, že se do návrhu vloudí chyba. Vítanou pomocí v tomto a podobných případech bude pojem nedeterministického konečného automatu, který nyní zavedeme.

Definice 2.1.2. *Nedeterministickým konečným automatem* budeme nazývat pětici $\mathscr{A} = (Q, \Sigma, \delta, I, F)$, kde Q a Σ jsou po řadě neprázdné konečné množiny stavů a vstupních symbolů,

$$\delta: Q \times \Sigma \to \mathscr{P}(Q)$$

je přechodová funkce [$\mathscr{P}(Q)$ označuje množinu všech podmnožin množiny Q], $I \subseteq Q$ je množina počátečních stavů a $F \subseteq Q$ je množina koncových stavů.

Pojem nedeterministického konečného automatu je zobecněním pojmu konečného automatu. Také tyto automaty lze analogicky reprezentovat tabulkou či stavovým diagramem. Na rozdíl od deterministického případu nemusí být u nedeterministických automatů stavem a vstupním symbolem jednoznačně určen stav, do kterého automat přejde. Je pouze vymezena množina stavů, do kterých může přejít. Druhým rozdílným rysem je obecně víceprvková množina počátečních stavů.

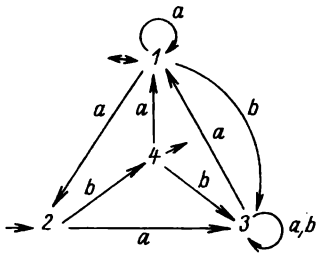
Příklad 2.1.3. Nedeterministický automat $\mathscr{A}$ reprezentovaný tabulkou

	a	b
$\leftrightarrow 1$	1, 2	3
$\rightarrow 2$	3	4
3	1, 3	3
$\leftarrow 4$	1	3

lze také reprezentovat stavovým diagramem na obr. 19.

Zobecnění automatu na nedeterministický automat vyžaduje příslušné rozšíření definice rozpoznávání jazyka.

Slovo je nedeterministickým konečným automatem přijímáno, jestliže odpovídá nějaké cestě z některého počátečního do některého koncového stavu. Tak automat z obr. 19 přijímá např. slovo *abaa*, protože odpovídá cestě, která prochází stavy 1, 1, 3, 1, 1 (nebo také cestám 1, 2, 4, 1, 1 a 2, 3, 3, 1, 1). Každá z uvedených cest začíná v počátečním a končí v koncovém stavu. Na přijetí slova *abaa* nic nemění skutečnost, že některé z cest, které mu odpovídají a začínají v počátečním stavu, v koncovém stavu nekončí. Příkladem je cesta 1, 2, 4, 1, 2. Podobně automat přijímá slovo *b*, protože může přejít ze stavu 2 do stavu 4. Žádná cesta odpovídající *b* a začínající v 1, tj. v druhém z počátečních stavů, přitom v koncovém stavu nekončí. Příkladem slov, která nejsou tímto automatem přijímána, jsou slova *bb* nebo *bab*. Tak slovu *bb* odpovídají dvě cesty (1, 3, 3 a 2, 4, 3) z počátečního stavu, ale žádná z nich nekončí v koncovém stavu.



Obr. 19

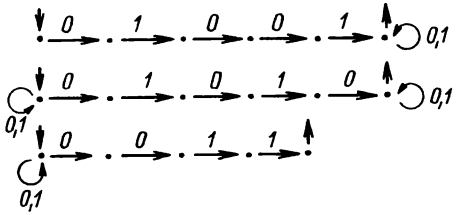
Definice 2.1.4. *Slovo* $w = x_1 \dots x_n \in \Sigma^+$ *je přijímáno nedeterministickým konečným automatem* $\mathscr{A} = (Q, \Sigma, \delta, I, F)$, jestliže existuje posloupnost $q_1, \dots, q_{n+1}$ stavů z Q taková, že $q_1 \in I$, $q_{n+1} \in F$ a pro všechna $i \in \{1, \dots, n\}$ je $q_{i+1} \in \delta(q_i, x_i)$. *Prázdné slovo je automatem přijímáno*, právě když $I \cap F \neq \emptyset$.

$L(\mathscr{A})$ označuje jazyk rozpoznávaný nedeterministickým konečným automatem $\mathscr{A}$, tj. množinu slov tímto automatem přijímaných.

Říkáme, že *jazyk L je rozpoznatelný nedeterministickým konečným automatem*, právě když existuje nedeterministický konečný automat $\mathscr{A}$ takový, že $L(\mathscr{A}) = L$.

Příklad 2.1.5. Jazyk z př. 2.1.1 je rozpoznáván nedeterministickým automatem z obr. 20.

Srovnejte, jak odpovídá struktura zadání jazyka v př. 2.1.1 se strukturou diagramu na obr. 20. Kromě toho si na obrázku po-



Obr. 20

všimněte ještě jednoho zajímavého rysu plynoucího z definice 2.1.4. Z některých vrcholů diagramu vychází méně než dvě šipky. To je zcela korektní, protože podle definice 2.1.2 přiřazuje přechodová funkce každé dvojici z $Q \times \Sigma$ jistou množinu pokračování. Také například prázdnou množinu. Pokud máte pochybnosti, jaký vliv má na přijetí slova výpočet, který takto uvázne v některém stavu, vraťte se k definici 2.1.4 a znovu si ji promyslete.

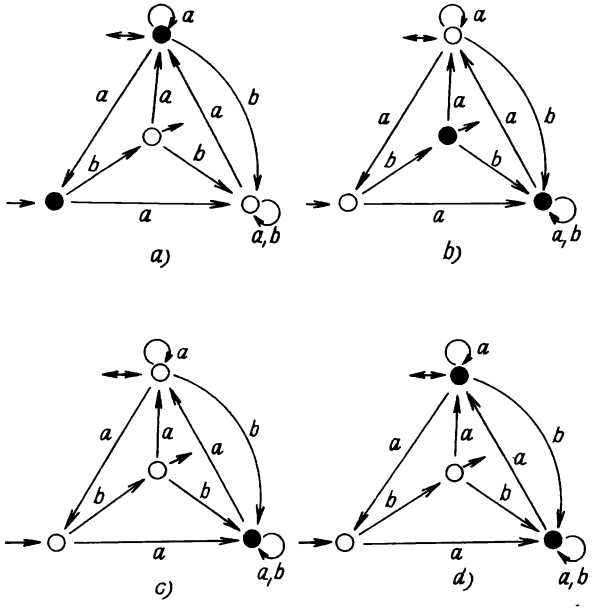
Na každý konečný automat lze pohlížet jako na speciální případ nedeterministického konečného automatu. Proto každý jazyk rozpoznatelný deterministickým konečným automatem je také rozpoznatelný automatem nedeterministickým. Důležité je, že platí i obrácené tvrzení.

Věta 2.1.6. *Každý jazyk rozpoznatelný nedeterministickým konečným automatem je také rozpoznatelný deterministickým konečným automatem.*

V důkazu věty se používá tzv. podmnožinové konstrukce. Tuto konstrukci si nejprve objasníme na intuitivní úrovni.

Chceme-li se přesvědčit, zda určitý nedeterministický konečný automat přijímá dané slovo, nemusí namátkové hledání v tabulce či stavovém diagramu vést k cíli a je třeba postupovat nějakým systematickým způsobem. Například si lze na stavovém diagramu automatu označit hracími kameny všechny stavy, do kterých se automat mohl na základě dosud přečteného úseku

vstupního slova dostat. Na obr. 21a) je zachycena výchozí pozice pro automat z př. 2.1.3, obr. 21b) zachycuje situaci po přečtení *b*, obr. 21c) situaci po přečtení *bb* a obr. 21d) situaci po přečtení *bba*. Okamžitě je vidět, že slovo *bba* je přijato, protože jeden z kamenů se ocitl ve vrcholu označujícím koncový stav.



Obr. 21

Od jedné pozice k druhé se přechází přesouváním kamenů podle příslušně ohodnocených šipek. Přitom někdy je třeba kameny přidávat, někdy ubírat [viz přechody c) → d) a b) → c) na obr. 21]. U každého jednotlivého automatu je možné vytvořit jen konečný počet vzájemně odlišných pozic (každá pozice je zcela popsána jistou podmnožinou stavů původního automatu – vrcholů, v nichž se právě nacházejí hrací kameny). Kromě toho je přechod od jedné pozice ke druhé jednoznačně určen čteným vstupním symbolem. Tato dvě pozorování ukazují, že systém, který jsme popsali, má základní vlastnosti charakteristické pro konečný automat, jak jsme o nich hovořili v úvodu první kapi-

toly. I počáteční stav a koncovou množinu lze snadno určit. Počátečním stavem je výchozí pozice, kdy hrací kameny jsou právě ve vrcholech odpovídajících počátečním stavům. Koncovým stavem našeho systému je každá pozice, u níž se alespoň jeden hrací kámen nachází v některém koncovém vrcholu.

Ke každému nedeterministickému konečnému automatu $\mathscr{A}$ tedy takto můžeme vytvořit systém, který je realizací jistého konečného automatu rozpoznávajícího $L(\mathscr{A})$. Tytéž myšlenky v precizní formě vyjadřuje formální důkaz věty, který nyní uvedeme.

Důkaz věty 2.1.6. Nechť $\mathscr{B} = (Q, \Sigma, \delta, I, F)$ je libovolný nedeterministický konečný automat. Sestrojíme konečný automat $\mathscr{A}$ rozpoznávající $L(\mathscr{B})$. Definujme funkci

$$\delta' : \mathscr{P}(Q) \times \Sigma \to \mathscr{P}(Q)$$

tímto předpisem: pro libovolné $K \in \mathscr{P}(Q)$ a $a \in \Sigma$ je

$$\delta'(K, a) = \bigcup_{q \in K} \delta(q, a) .$$

Dále definujme

$$Q' = \mathscr{P}(Q), \; q_0 = I, \; F' = \{K \in \mathscr{P}(Q); K \cap F \neq \emptyset\} .$$

Tvrdíme, že automat

$$\mathscr{A} = (Q', \Sigma, \delta', q_0, F')$$

rozpoznává $L(\mathscr{B})$. To nyní ověříme. Nejprve si uvědomme, že

$$e \in L(\mathscr{A}) \Leftrightarrow I \in F' \Leftrightarrow I \cap F \neq \emptyset \Leftrightarrow e \in L(\mathscr{B}) .$$

Případ prázdného slova je tím vyřízen a zbývá ověřit, že pro každé neprázdné slovo w je

$$w \in L(\mathscr{B}) \Leftrightarrow w \in L(\mathscr{A}) .$$

1. Předpokládejme, že

$$w = a_1 \dots a_n \in L(\mathscr{A}), \; n \geqq 1 .$$

Pak existují

$$K_1, \dots, K_{n+1} \in \mathscr{P}(Q)$$

taková, že

$$K_1 = I\,,\ K_{n+1} \in F'$$

a pro všechna i, $1 \leqq i \leqq n$, je

$$K_{i+1} = \delta(K_i, a_i)\,.$$

Z toho odvodíme, že pro každé r takové, že $0 \leqq r \leqq n-1$ existuje posloupnost

$$q_{n-r}, q_{n-r+1}, \ldots, q_{n+1}\,,$$

která splňuje tyto podmínky:

$$q_{n+1} \in K_{n+1}\,,\ q_{n+1} \in F\,,$$

$$q_{i+1} \in \delta(q_i, a_i)\,,\ q_i \in K_i$$

pro $n \geqq i \geqq n-r$. Důkaz provedeme indukcí podle r.

I. $r = 0$. Máme ukázat, že existují q_n, q_{n+1} taková, že

$$q_n \in K_n\,,\ q_{n+1} \in K_{n+1} \cap F \text{ a } q_{n+1} \in \delta(q_n, a_n)\,.$$

Víme, že $K_{n+1} \in F'$. Odtud plyne, že $K_{n+1} \cap F \neq \emptyset$. Zvolme libovolně

$$q_{n+1} \in K_{n+1} \cap F\,.$$

Protože

$$q_{n+1} \in K_{n+1} = \delta(K_n, a_n)\,,$$

existuje $q_n \in K_n$ takové, že

$$q_{n+1} \in \delta(q_n, a_n)\,.$$

II. Předpokládejme, že je tvrzení dokázáno pro nějaké r splňující podmínku

$$0 \leqq r \leqq n-1\,,$$

tzn. existuje posloupnost

$$q_{n-r}, \ldots, q_{n+1}$$

požadovaných vlastností. K tomu, abychom dokázali, že tvrzení platí také pro $r + 1$, stačí ukázat, že existuje

$$q_{n-r-1} \in K_{n-r-1}$$

takové, že

$$q_{n-r} \in \delta(q_{n-r-1}, a_{n-r-1}).$$

To ovšem plyne ze skutečnosti, že

$$q_{n-r} \in K_{n-r}$$

a

$$K_{n-r} = \delta(K_{n-r-1}, a_{n-r-1}) = \bigcup_{q \in K_{n-r-1}} \delta(q, a_{n-r-1}).$$

Tím je tvrzení dokázáno pro všechna r, $0 \leqq r \leqq n - 1$, speciálně pro $r = n - 1$.

Tedy existuje posloupnost $q_1, \ldots, q_{n+1}$ požadovaných vlastností. Z toho okamžitě vyplývá, že $a_1 \ldots a_n = w \in L(\mathscr{B})$. Tím jsme ověřili, že $L(\mathscr{A}) \subseteq L(\mathscr{B})$.

2. Budiž

$$w = a_1 \ldots a_n \in L(\mathscr{B}).$$

Existuje tedy posloupnost $q_1, \ldots, q_{n+1}$ stavů z Q splňující tyto podmínky:

(i) $q_1 \in I$,

(ii) $q_{i+1} \in \delta(q_i, a_i)$ pro všechna i, $1 \leqq i \leqq n$,

(iii) $q_{n+1} \in F$.

Definujme posloupnost $K_1, \ldots, K_{n+1}$ předpisem

$$K_1 = I, \; K_{i+1} = \delta'(K_i, a_i)$$

pro $1 \leqq i \leqq n$.

Podle podmínky (i) je $q_1 \in I$, podmínka (ii) zaručuje, že $q_j \in K_j$ pro všechna j, $1 \leqq j \leqq n + 1$, jak se snadno ověří indukcí. Podle (iii) je proto

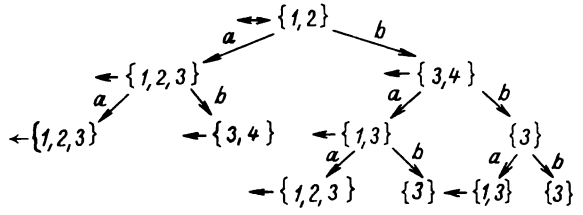
$$K_{n+1} \cap F \neq \emptyset.$$

To znamená, že $K_{n+1} \in F'$ a $a_1 \ldots a_n \in L(\mathscr{A})$.

Poznámka 2.1.7. Důkaz, který jsme právě podali, je konstruktivní stejně jako ostatní důkazy v této knize. To znamená, že jsme nejenom dokázali, že ke každému nedeterministickému konečnému automatu existuje s ním ekvivalentní deterministický konečný automat, ale zároveň jsme získali algoritmus, kterým lze příslušný automat sestrojit.

Poznámka 2.1.8. Popsaný postup vede od nedeterministického automatu o n stavech k deterministickému automatu o 2^n stavech. Tomuto enormnímu vzrůstu velikosti se často můžeme vyhnout tím, že okamžitě vyloučíme stavy, které nejsou dosažitelné z počátečního stavu. Většinou přímo konstruujeme stavový strom příslušného deterministického automatu tak, jak to ukážeme v následujícím příkladu. Výsledný automat lze často dále zmenšit algoritmem redukce popsaným v čl. 1.5. Výsledkem je vůbec nejmenší možný deterministický konečný automat ekvivalentní s výchozím automatem. Přechod od výchozího nedeterministického automatu k výslednému minimálnímu deterministickému automatu je čistě mechanický a lze jej např. realizovat na počítači.

Příklad 2.1.9. K nedeterministickému automatu z př. 2.1.3 sestrojme ekvivalentní deterministický konečný automat. Konstruujeme přímo stavový strom automatu. Výsledek je na obr. 22. Dostáváme pouze 5 stavů, zatímco $\mathscr{P}(Q)$, množina všech podmnožin množiny Q, má 16 prvků. Redukt tohoto automatu má dokonce jen 4 stavy (viz čl. 1.5).



Obr. 22

Poznámka 2.1.10. V některých případech se může stát, že výsledný minimální automat bude mít stále 2^n stavů oproti n stavům výchozího nedeterministického automatu. (Podrobněji o tom po-

jednáme v čl. 2.6.) V takovém případě je už jen pro trochu větší n nerealistické provádět návrh deterministického automatu pro reprezentaci příslušného jazyka. Přesněji řečeno, může být prakticky neproveditelné sestrojit tabulku nebo graf deterministického automatu. Avšak výchozí nedeterministický automat může velmi dobře posloužit při praktické realizaci deterministického automatu způsobem, který jsme naznačili při intuitivním zavedení podmnožinové konstrukce (za větou 2.1.6). Jestliže výchozí nedeterministický automat $\mathscr{B}$ má n stavů, potom základem realizace ekvivalentního konečného automatu je n-bitová paměť. Každý bit odpovídá jednomu stavu automatu $\mathscr{B}$. Množina bitů této paměti obsahujících jedničku vždy charakterizuje nějakou podmnožinu stavového prostoru automatu $\mathscr{B}$ (analogicky podmnožině vrcholů s hracími kameny). K aktualizaci stavu paměti v závislosti na dalším vstupním symbolu pak stačí vhodná booleovská funkce. Tento postup lze využít jak k softwarové, tak hardwarové realizaci konečných automatů.

Pojem nedeterministického konečného automatu je užitečné dále zobecnit zavedením tzv. e-přechodů.

Definice 2.1.11. Definujme zobecněný nedeterministický konečný automat jako pětici $\mathscr{A} = (Q, \Sigma, \delta, I, F)$, kde Q, Σ, I, F jsou po řadě konečné množiny stavů, vstupních symbolů, počátečních a koncových stavů a δ je přechodová funkce

$$\delta: Q \times (\Sigma \cup \{e\}) \to \mathscr{P}(Q).$$

Oproti nedeterministickému konečnému automatu zde přechodová funkce připouští *e-přechody*, tzn. možnost přejít z jednoho stavu do druhého, aniž by se přečetl další vstupní symbol.

Definujme relaci $\vdash^*_{\mathscr{A}}$ na $(Q \times \Sigma^*) \times Q$ [$(q, u) \vdash^*_{\mathscr{A}} q'$ má tento význam: slovo u může převést automat $\mathscr{A}$ ze stavu q do stavu q'] indukcí podle délky slov ze Σ^* takto:

1. $(q, e) \vdash^*_{\mathscr{A}} q$

pro všechna q,

$(q, e) \vdash^*_{\mathscr{A}} q'$

pro všechna q, q' taková, že $q' \in \delta(q, e)$;

2. $(q, a) \vdash_{\mathscr{A}}^{*} q'$

pro všechna $q, q' \in Q$ a $a \in \Sigma$ taková, že $q' \in \delta(q, a)$;

3. $(q, uv) \vdash_{\mathscr{A}}^{*} q'$,

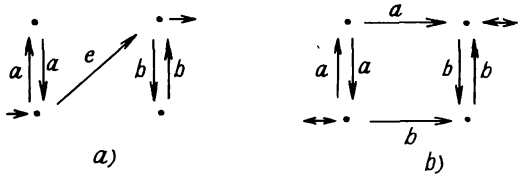
právě když existuje $\tilde{q} \in Q$ takové, že

$$(q, u) \vdash_{\mathscr{A}}^{*} \tilde{q} \text{ a } (\tilde{q}, v) \vdash_{\mathscr{A}}^{*} q' \quad (u, v \in \Sigma^{*}).$$

Jazyk $L(\mathscr{A})$ rozpoznávaný zobecněným nedeterministickým konečným automatem $\mathscr{A}$ je pak definován takto:

$$L(\mathscr{A}) = \{w; (\exists s \in I)(\exists r \in F)(s, w) \vdash_{\mathscr{A}}^{*} r\}.$$

Příklad 2.1.12. Zobecněné nedeterministické konečné automaty je možné zcela analogicky jako nedeterministické konečné automaty reprezentovat stavovým diagramem. Jednoduchý příklad takového diagramu je na obr. 23a). Tento automat zřejmě přijímá slova tvořená sudým počtem symbolů a následovaných sudým počtem symbolů b.



Obr. 23

Věta 2.1.13. *Každý jazyk rozpoznatelný zobecněným nedeterministickým konečným automatem je rozpoznatelný konečným automatem.*

Důkaz. Budiž $\mathscr{A} = (Q, \Sigma, \delta, I, F)$ zobecněný nedeterministický konečný automat. Podle věty 2.1.6 stačí dokázat, že $L(\mathscr{A})$ je rozpoznatelný jistým nedeterministickým automatem $\tilde{\mathscr{A}}$. Sestrojíme takový automat

$$\tilde{\mathscr{A}} = (\tilde{Q}, \Sigma, \tilde{\delta}, \tilde{I}, \tilde{F}).$$

Položíme

$$\tilde{Q} = Q\,,\quad \tilde{I} = \{\tilde{s} \in Q;\, (\exists s \in I)\,[(s, e) \vdash^*_{\mathscr{A}} \tilde{s}]\}\,,$$

$$\tilde{F} = \{\tilde{r} \in Q;\, (\exists r \in F)\,[(\tilde{r}, e) \vdash^*_{\mathscr{A}} r]\}$$

a funkci $\tilde{\delta}$ definujeme předpisem

$$q' \in \tilde{\delta}(q, a) \Leftrightarrow_{\mathrm{df}} (q, a) \vdash^*_{\mathscr{A}} q'$$

pro každé $q, q' \in Q$, $a \in \Sigma$.

Potom $L(\mathscr{A}) = L(\tilde{\mathscr{A}})$, neboť

1. $e \in L(\mathscr{A}) \Leftrightarrow (s, e) \vdash^*_{\mathscr{A}} r$ pro nějaké $s \in I$, $r \in F \Leftrightarrow$
$\Leftrightarrow \tilde{I} \cap \tilde{F} \neq \emptyset \Leftrightarrow e \in L(\tilde{\mathscr{A}})$.

2. Pro $n \geqq 1$, $a_1, \ldots, a_n \in \Sigma$:

$a_1 \ldots a_n \in L(\mathscr{A}) \Leftrightarrow$
$\Leftrightarrow (s, a_1 \ldots a_n) \vdash^*_{\mathscr{A}} r$ pro nějaké $s \in I$, $r \in F \Leftrightarrow$
$\Leftrightarrow$ pro nějaká $q_1, \ldots, q_n \in Q$, $s \in I$, $r \in F$ je

$(s, a_1) \vdash^*_{\mathscr{A}} q_1 \,\&\, (q_1, a_2) \vdash^*_{\mathscr{A}} q_2 \,\&\, \ldots \,\&\, (q_{n-1}, a_n) \vdash^*_{\mathscr{A}} r \Leftrightarrow$
$\Leftrightarrow$ pro nějaká $q_1, \ldots, q_n \in Q$, $s \in \tilde{I}$, $r \in \tilde{F}$ je

$q_1 \in \tilde{\delta}(s, a_1) \,\&\, q_2 \in \tilde{\delta}(q_1, a_2) \,\&\, \ldots \,\&\, r \in \tilde{\delta}(q_{n-1}, a_n) \Leftrightarrow$
$\Leftrightarrow a_1 \ldots a_n \in L(\tilde{\mathscr{A}})$.

Příklad 2.1.14. Na obr. 23b) je nedeterministický konečný automat sestrojený k automatu z obr. 23a) konstrukcí z důkazu předchozí věty.

2.2. Uzávěrové vlastnosti

Návrh automatu pro rozpoznávání určitého jazyka lze někdy zjednodušit. Často se totiž daný jazyk podaří vyjádřit pomocí několika jednodušších jazyků. Potom je možné navrhnout automaty pro realizaci jednodušších jazyků a na jejich základě sestrojit výsledný automat. Důležitou pomoc zde poskytují teoretické poznatky o uzávěrových vlastnostech třídy $\mathscr{F}$ jazyků rozpoznatelných konečnými automaty.

Nejprve se seznámíme s několika důležitými operacemi nad jazyky. V dalších odstavcích budeme předpokládat, že L, L_1 a L_2 jsou libovolné jazyky nad jistou abecedou Σ a $w \in \Sigma^*$.

Jazyky jsou množiny slov, proto s nimi lze provádět základní množinové operace:

sjednocení

$$L_1 \cup L_2$$

je jazyk obsahující právě všechna slova náležející buď do L_1, nebo do L_2, nebo do obou současně;

průnik

$$L_1 \cap L_2$$

je jazyk obsahující právě všechna slova náležející současně do L_1 i do L_2;

doplněk

$$\Sigma^* - L$$

obsahuje právě všechna slova z Σ^* nepatřící do L. Pokud bude ze souvislosti jasné, o jaké Σ^* se jedná, budeme místo $\Sigma^* - L$ psát jen $-L$;

rozdíl

$$L_1 - L_2$$

obsahuje právě všechna slova z L_1 nepatřící do L_2.

Připomeňme známé množinové identity, které budeme nejčastěji potřebovat

$$\left.\begin{aligned} L_1 \cap L_2 &= -(-L_1 \cup -L_2) \\ L_1 \cup L_2 &= -(-L_1 \cap -L_2) \end{aligned}\right\} \text{ de Morganova pravidla,}$$

$$L_1 - L_2 = L_1 \cap (\Sigma^* - L_2).$$

Příklad 2.2.1. Jazyk

$$L = \{w \in \{0, 1\}^*;$$
$$w \text{ obsahuje sudý počet nul a každá jednička je}$$
$$\text{bezprostředně následována alespoň jednou nulou}\}$$

můžeme vyjádřit množinovými operacemi pomocí tří jednodušeji charakterizovaných jazyků takto:

$$L = L_1 - (L_2 \cup L_3),$$

kde

$$L_1 = \{w \in \{0,1\}^*;\ w \text{ obsahuje sudý počet nul}\},$$

$$L_2 = \{w \in \{0,1\}^*;\ w \text{ obsahuje podslovo } 11\},$$

$$L_3 = \{w \in \{0,1\}^*;\ w \text{ končí jedničkou}\}.$$

Další operace, kterými se budeme zabývat, jsou už specifické pro jazyky.

Definice 2.2.2. 1. *Součinem* (nebo též *zřetězením*) *jazyků* L_1 a L_2 nazýváme jazyk

$$L_1 . L_2 = \{uv;\ u \in L_1 \ \& \ v \in L_2\}.$$

2. *n-tou mocninu* L^n *jazyka* L definujeme induktivně takto:

$$L^0 =_{\text{df}} \{e\},$$

$$L^{n+1} =_{\text{df}} L^n . L \quad \text{pro každé } n \geqq 0.$$

3. *Iteraci* L^* *jazyka* L a *pozitivní iteraci* L^+ *jazyka* L definujeme takto:

$$L^+ =_{\text{df}} L \cup L^2 \cup L^3 \cup \ldots = \bigcup_{i=1}^{\infty} L^i,$$

$$L^* =_{\text{df}} L^0 \cup L \cup L^2 \cup L^3 \cup \ldots = \bigcup_{i=0}^{\infty} L^i.$$

Příklad 2.2.3. Označme

$$L_1 = \{ab^i;\ i \geqq 0\},\ L_2 = \{c, cc\}.$$

Potom

$$L_1 . L_2 = \{ab^i c^j;\ i \geqq 0 \ \& \ 1 \leqq j \leqq 2\},$$

$$L_1^+ = \{ab^{i_1} ab^{i_2} \ldots ab^{i_k};\ k \geqq 1 \ \& \ 0 \leqq i_j$$

pro každé j, $1 \leqq j \leqq k\}$,

$$L_1^* = L_1^+ \cup \{e\}.$$

Všimněte si, že definice 1.2.1 množin Σ^+ a Σ^* je v souladu se zavedením operací $^+$ a *. Dále zřejmě platí, že

$$L \,.\, \emptyset = \emptyset \,.\, L = \emptyset$$

a

$$\{e\} \,.\, L = L \,.\, \{e\} = L$$

($\{e\}$ je jazyk obsahující jediné slovo, totiž prázdné slovo). Je zřejmý i vztah

$$L^* = L^+ \cup \{e\} \,.$$

Dohodněme se ještě, že tam, kde to nepovede k nedorozumění, budeme $\{w\}$ ztotožňovat s w a psát např. $L - e$, nebo wL, Lw.

Definice 2.2.4. 1. *Levý kvocient jazyka* L_1 *podle jazyka* L_2 je jazyk

$$L_2 \backslash L_1 = \{u;\ wu \in L_1 \text{ pro nějaké } w \in L_2\} \,.$$

2. *Pravý kvocient jazyka* L_1 *podle jazyka* L_2 je jazyk

$$L_1 / L_2 = \{u;\ uw \in L_1 \text{ pro nějaké } w \in L_2\} \,.$$

3. *Levá derivace jazyka* L *podle slova* w je jazyk

$$\partial_w L = w \backslash L \,.$$

4. *Pravá derivace jazyka* L *podle slova* w je jazyk

$$\partial^r_w L = L / w \,.$$

Levý kvocient $L_2 \backslash L_1$ je tedy tvořen všemi slovy, která vzniknou ze slov L_1 odtržením nějakého počátečního úseku patřícího do L_2. Podobně L_1 / L_2 obsahuje slova vzniklá ze slov L_1 po odtržení nějakých jejich koncových úseků patřících do L_2.

Příklad 2.2.5. Definujme

$$L_1 = \{a^i b a^j b a^i b a^j;\ i, j \geqq 1\} \,,$$
$$L_2 = \{a^i b a^i b;\ i \geqq 1\} \,,$$
$$L_3 = \{b a^{2j};\ j \geqq 5\} \,.$$

Potom

$$L_2 \backslash L_1 = \{a^i b a^i;\ i \geqq 1\} \,,$$
$$L_1 / L_3 = \{a^i b a^{2j} b a^i;\ i \geqq 1, j \geqq 5\} \,.$$

Definice 2.2.6. *Inverzí slova* $w = a_1 \dots a_n$ budeme nazývat slovo $w^R = a_n \dots a_2 a_1$, speciálně $e^R = e$. *Inverzí jazyka* L budeme nazývat jazyk $L^R = \{w^R; w \in L\}$.

Definice 2.2.7. Nechť Σ je konečná abeceda a pro každé $a \in \Sigma$ budiž $\sigma(a)$ jazyk v nějaké abecedě Δ_a. Dále položme

$$\sigma(e) = e$$

a

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

pro každé $u, v \in \Sigma^*$. Potom zobrazení

$$\sigma: \Sigma^* \to \mathscr{P}(\Delta^*),$$

kde

$$\Delta = \bigcup_{a \in \Sigma} \Delta_a,$$

se nazývá *substituce*. Pro každý jazyk $L \subseteq \Sigma^*$ pak definujeme

$$\sigma(L) =_{\text{df}} \bigcup_{w \in L} \sigma(w)$$

a říkáme, že *jazyk* $\sigma(L)$ *vznikl z jazyka* L *substitucí* σ. Substituci σ nazýváme *nevypouštějící*, jestliže žádný z jazyků $\sigma(a)$, $a \in \Sigma$, neobsahuje prázdné slovo.

Příklad 2.2.8. Pro abecedu $\Sigma = \{0, 1\}$ definujme substituci σ předpisem

$$\sigma(0) = \{a^i b^i; i \geqq 1\},$$

$$\sigma(1) = \{cd, cdd\}.$$

Tato substituce je zobrazení

$$\sigma: \Sigma^* \to \mathscr{P}(\Delta^*),$$

tzn. každému slovu v abecedě Σ přiřazuje jazyk v abecedě Δ, kde

$$\Delta = \{a, b\} \cup \{c, d\} = \{a, b, c, d\}.$$

Například

$$\sigma(010) = \sigma(0).\sigma(10) = \sigma(0).\sigma(1).\sigma(0) =$$
$$= \{a^i b^i;\ i \geqq 1\}.\{cd, cdd\}.\{a^i b^i;\ i \geqq 1\} =$$
$$= \{a^i b^i cda^j b^j;\ i, j \geqq 1\} \cup$$
$$\cup \{a^i b^i cdda^j b^j;\ i, j \geqq 1\}.$$

Pro jazyk

$$L = \{10^m 1;\ m \geqq 2\}$$

bude

$$\sigma(L) = \{cd^k a^{i_1} b^{i_1} a^{i_2} b^{i_2} \ldots a^{i_m} b^{i_m} cd^l;\ m \geqq 2 \ \&\ i_j \geqq 1$$

pro všechna $1 \leqq j \leqq m \ \&\ k, l \in \{1, 2\}\}$.

Tato substituce je nevypouštějící, protože $\sigma(0)$ ani $\sigma(1)$ neobsahují prázdné slovo.

Definice 2.2.9. (Značení jako v předchozí definici.) Substituce σ, u níž pro každý symbol $a \in \Sigma$ obsahuje $\sigma(a)$ jediné slovo w_a, se nazývá *homomorfismus*. Homomorfismus lze tedy považovat za zobrazení $\sigma: \Sigma^* \to \Delta^*$. Jestliže $\sigma(a) \neq \{e\}$ pro každé $a \in \Sigma$, nazývá se σ *nevypouštějící homomorfismus*.

Příklad 2.2.10. Definujme nevypouštějící homomorfismus

$$h: \{0, 1\}^* \to \{a, b, c\}^*$$

předpisem

$$h(0) = bcca,\ h(1) = aa.$$

Potom např.

$$h(0101) = bccaaabccaaa$$

a pro

$$L = \{01^i 0;\ i \geqq 1\}$$

je

$$h(L) = \{bcca^{2k+1} bcca;\ k \geqq 1\}.$$

Už z uvedených příkladů lze odhadnout, že i značně komplikované jazyky je někdy možné vyjádřit jako výsledek několika

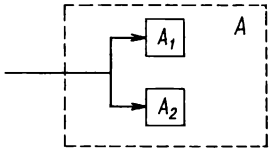
základních operací provedených nad jazyky poměrně jednoduchými.

Toto konstatování je důležité zejména ve spojení se skutečností, že uvedené typy operací vedou od jazyků z třídy všech jazyků rozpoznatelných konečnými automaty $\mathscr{F}$ opět jen k jazykům patřícím do $\mathscr{F}$.

Věta 2.2.11. *Pro libovolné jazyky* $L_1, L_2 \subseteq \Sigma^*$ *platí toto tvrzení: jestliže* $L_1, L_2 \in \mathscr{F}$, *potom také* $L_1 \cap L_2$, $L_1 \cup L_2$, $\Sigma^* - L_1 \in \mathscr{F}$.

Jinými slovy: *Třída jazyků rozpoznatelných konečnými automaty je uzavřena vůči operacím průniku, sjednocení a doplňku.*

Než přikročíme k formálnímu důkazu věty, věnujme se alespoň chvíli intuitivní představě, z níž tento důkaz vychází.



Obr. 24

Víme-li, že $L_1, L_2 \in \mathscr{F}$, znamená to, že existují dva konečné deterministické automaty $\mathscr{A}_1$ a $\mathscr{A}_2$ rozpoznávající po řadě jazyky L_1 a L_2. Z nich snadno vytvoříme systém rozpoznávající jazyk $L_1 \cap L_2$. Stačí „spouštět oba automaty současně" (viz obr. 24) a oběma předkládat stejné vstupní slovo. Toto slovo patří do $L_1 \cap L_2$, jestliže se po jeho přečtení oba automaty dostanou do koncového stavu. Vytvořený systém zřejmě má vlastnosti, které jsme v kap. 1 označili jako podstatné pro konečný automat. Může se dostávat jen do konečně mnoha stavů: každý jeho stav je určen nějakou dvojicí (p_1, p_2) určující stav obou jeho složek, tj. stav p_1 automatu $\mathscr{A}_1$ a stav p_2 automatu $\mathscr{A}_2$. Vstupní abeceda systému je stejná jako vstupní abeceda automatů $\mathscr{A}_1$ a $\mathscr{A}_2$, počáteční stav systému je stav, v němž obě složky systému jsou v počátečních stavech, obdobně pro koncové stavy. Můžeme tedy na celý systém $\mathscr{A}$ (obr. 24) pohlížet jako na konečný automat rozpoznávající $L_1 \cap L_2$. Podobnou úvahou můžeme dojít k uzavřenosti $\mathscr{F}$ vůči sjednocení a doplňku. Následující důkaz je přesným matematickým vyjádřením těchto představ.

Důkaz věty 2.2.11. Nechť L_1 je rozpoznáván konečným automatem

$$\mathscr{A}_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$$

a L_2 automatem

$$\mathscr{A}_2 = (Q_2, \Sigma, \delta_2, q_2, F_2).$$

1. Jazyk $L_1 \cap L_2$ je pak rozpoznáván konečným automatem

$$\mathscr{A} = (Q, \Sigma, \delta, q_0, F),$$

kde

$$Q = Q_1 \times Q_2, \quad q_0 = (q_1, q_2), \quad F = F_1 \times F_2$$

a δ je definována předpisem: pro libovolné

$$(p_1, p_2) \in Q, \quad a \in \Sigma$$

je

$$\delta((p_1, p_2), a) = (\delta_1(p_1, a), \delta_2(p_2, a)).$$

$\mathscr{A}$ je konečný automat a ukážeme, že rozpoznává $L_1 \cap L_2$. Pro libovolné $w \in \Sigma^*$ je

$$\delta((q_1, q_2), w) = (\delta_1(q_1, w), \delta_2(q_2, w)),$$

jak se snadno ověří indukcí. Tedy

$$\begin{aligned} w \in L_1 \cap L_2 &\Leftrightarrow w \in L_1 \;\&\; w \in L_2 \Leftrightarrow \\ &\Leftrightarrow \delta_1(q_1, w) \in F_1 \;\&\; \delta_2(q_2, w) \in F_2 \Leftrightarrow \\ &\Leftrightarrow (\delta_1(q_1, w), \delta_2(q_2, w)) \in F_1 \times F_2 \Leftrightarrow \\ &\Leftrightarrow \delta(q_0, w) \in F. \end{aligned}$$

Proto skutečně

$$L(\mathscr{A}) = L(\mathscr{A}_1) \cap L(\mathscr{A}_2).$$

2. Jazyk $\Sigma^* - L_1$ je zřejmě rozpoznáván automatem

$$(Q_1, \Sigma, \delta_1, q_1, Q_1 - F_1).$$

3. Pro důkaz tvrzení, že také $L_1 \cup L_2 \in \mathscr{F}$, se nabízí několik možností. Jednak je možné vytvořit automat rozpoznávající

$L_1 \cup L_2$ přímo konstrukcí podobnou jako v části 1, pouze jako množinu koncových stavů je v tomto případě třeba vzít

$$(F_1 \times Q_2) \cup (Q_1 \times F_2).$$

Druhá možnost je využít důkazu částí 1 a 2 takto: předpokládáme, že

$$L_1, L_2 \in \mathscr{F}.$$

Podle 2 je také

$$-L_1, -L_2 \in \mathscr{F}.$$

Proto i

$$-L_1 \cap -L_2 \in \mathscr{F}$$

podle 1 a konečně

$$-(-L_1 \cap -L_2) \in \mathscr{F}$$

opět podle 2. Jedno z de Morganových pravidel ale dává

$$-(-L_1 \cap -L_2) = L_1 \cup L_2,$$

takže

$$L_1 \cup L_2 \in \mathscr{F}.$$

Mezi oběma cestami není podstatný rozdíl, neboť vedou ke konstrukci téhož automatu. Poslední možnost, kterou uvedeme, se opírá o pojem nedeterministického automatu. Vyjdeme z obecnějšího předpokladu, že jazyky L_1, L_2 jsou po řadě reprezentovány nedeterministickými konečnými automaty

$$\mathscr{A}_1 = (Q_1, \Sigma, \delta_1, I_1, F_1)$$

a

$$\mathscr{A}_2 = (Q_2, \Sigma, \delta_2, I_2, F_2).$$

Ukážeme, jak k $\mathscr{A}_1$, $\mathscr{A}_2$ lze sestrojit nedeterministický konečný automat

$$\mathscr{A} = (Q, \Sigma, \delta, I, F)$$

takový, že

$$L(\mathscr{A}) = L_1 \cup L_2.$$

Bez újmy na obecnosti můžeme předpokládat, že

$$Q_1 \cap Q_2 = \emptyset .$$

(Toho lze dosáhnout vhodným přejmenováním stavů.) Potom stačí položit

$$Q = Q_1 \cup Q_2 , \; I = I_1 \cup I_2 , \; F = F_1 \cup F_2 .$$

Přechodovou funkci definujme tímto předpisem:

$$\delta(q, a) = \delta_1(q, a), \text{ pokud } q \in Q_1 \; \& \; a \in \Sigma ,$$

$$\delta(q, a) = \delta_2(q, a), \text{ pokud } q \in Q_2 \; \& \; a \in \Sigma .$$

Je zřejmé, že $\mathscr{A}$ skutečně rozpoznává $L_1 \cup L_2$.

První způsob vedl od výchozích deterministických automatů opět k automatu deterministickému. Naposledy popsaná konstrukce vede i v případě, že výchozí automaty jsou deterministické, k automatu nedeterministickému (s víceprvkovou počáteční množinou). Její praktický význam spočívá jednak v širší použitelnosti, jednak v tom, že zkonstruovaný automat má $|Q_1| + |Q_2|$ stavů na rozdíl od $|Q_1| . |Q_2|$ stavů automatu z první konstrukce.

Příklad 2.2.12. Sestrojme automat rozpoznávající jazyk

$$L = \{w \in \{0, 1\}^*;$$
ve w buď není počet nul dělitelný třemi,
nebo počet jedniček není dělitelný čtyřmi$\}$.

Položíme

$$L_1 = \{w \in \{0, 1\}^*;$$
ve w je počet nul dělitelný třemi$\}$,

$$L_2 = \{w \in \{0, 1\}^*;$$
ve w je počet jedniček dělitelný čtyřmi$\}$.

Potom $L = -L_1 \cup -L_2$. Konstrukce příslušného automatu proto může postupovat po krocích takto: jazyk L_1 je rozpoznáván automatem z obr. 25a), jazyk L_2 automatem z obr. 25b),

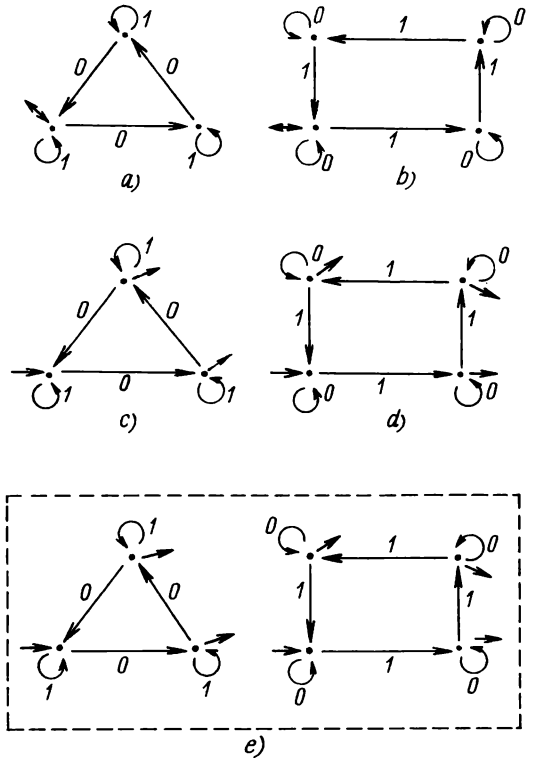
jazyk $-L_1$ automatem z obr. 25c), jazyk $-L_2$ automatem z obr. 25d) a konečně $-L_1 \cup -L_2$ automatem z obr. 25e).

Uzávěrových vlastností třídy $\mathscr{F}$ lze často s výhodou využít při ověřování, že určitý jazyk není rozpoznatelný konečným automatem.

Příklad 2.2.13. Ukážeme, že jazyk

$$L = \{w \in \{0,1\}^*;$$
$$w \text{ obsahuje stejný počet nul a jedniček}\}$$

není rozpoznatelný konečným automatem.



Obr. 25

Důkaz by bylo možné provést přímým užitím věty 1.3.3 jako v př. 1.3.6. Jestliže už ale víme, že jazyk

$$L = \{0^i 1^i;\ i \geqq 1\}$$

není rozpoznatelný konečným automatem, lze důkaz vést jednodušeji.

Předpokládejme, že L je rozpoznatelný konečným automatem. Protože $L_1 = \{0^i 1^j;\ i, j \geqq 1\}$ je rozpoznatelný konečným automatem (sestrojte!), je podle věty 2.2.11 také $L_1 \cap L_2$ rozpoznatelný konečným automatem. Ale

$$L_1 \cap L_2 = \{0^n 1^n;\ n \geqq 1\}.$$

To je spor s 1.3.6, a proto L není rozpoznatelný konečným automatem.

Čtenář, který si přečetl čl. 1.5 o redukci konečných automatů, zná efektivní metodu, jak pro libovolné dva automaty $\mathscr{A}_1$, $\mathscr{A}_2$ rozhodnout, zda $L(\mathscr{A}_1) = L(\mathscr{A}_2)$. K jiné metodě, i když ne tak efektivní, se dostaneme využitím uzávěrových vlastností.

Věta 2.2.14. *Existuje algoritmus, který pro libovolné dva konečné automaty $\mathscr{A}_1$, $\mathscr{A}_2$ rozhoduje, zda $L(\mathscr{A}_1) = L(\mathscr{A}_2)$.*

Důkaz. Nejprve si uvědomme, že pro každý konečný automat jsme schopni rozhodnout, zda $L(\mathscr{A}) \neq \emptyset$. Stačí se např. přesvědčit, zda v diagramu automatu $\mathscr{A}$ existuje orientovaná cesta (i délky 0) z počátečního stavu do některého koncového stavu.

Dále je třeba vzít v úvahu, že

$$L(\mathscr{A}_1) = L(\mathscr{A}_2),$$

právě když

$$(L(\mathscr{A}_1) - L(\mathscr{A}_2)) \cup (L(\mathscr{A}_2) - L(\mathscr{A}_1)) = \emptyset.$$

Podle 2.2.11 lze z $\mathscr{A}_1$ a $\mathscr{A}_2$ sestrojit konečný automat $\mathscr{A}$ tak, že

$$L(\mathscr{A}) = (L(\mathscr{A}_1) - L(\mathscr{A}_2)) \cup (L(\mathscr{A}_2) - L(\mathscr{A}_1)).$$

Stačí pak ověřit, zda

$$L(\mathscr{A}) = \emptyset.$$

Stejné oblasti použití, jaké ilustrovaly př. 2.2.12 a 2.2.13, mají i ostatní uzávěrové vlastnosti třídy $\mathscr{F}$.

Věta 2.2.15. *Pro libovolné dva jazyky* $L_1, L_2 \in \mathscr{F}$ *je také* $L_1 . L_2 \in \mathscr{F}$ *a* $L_1^* \in \mathscr{F}$.

Důkaz. Buďte

$$\mathscr{A}_1 = (Q_1, \Sigma, \delta_1, I_1, F_1),$$

$$\mathscr{A}_2 = (Q_2, \Sigma, \delta_2, I_2, F_2)$$

nedeterministické konečné automaty reprezentující po řadě jazyky L_1 a L_2. Předpokládáme-li, že výchozí automaty jsou obecně nedeterministické, tím spíše budou platit následující konstrukce a úvahy pro jejich speciální případ, automaty deterministické. Případným přejmenováním stavů lze docílit toho, aby

$$Q_1 \cap Q_2 = \emptyset .$$

1. Sestrojíme nedeterministický automat

$$\mathscr{A} = (Q, \Sigma, \delta, I, F)$$

takový, že

$$L(\mathscr{A}) = L_1 . L_2 .$$

Položme

$$Q = Q_1 \cup Q_2 , \quad F = F_2 ,$$

$$I = \begin{cases} I_1, \text{ jestliže } I_1 \cap F_1 = \emptyset, \text{ tzn. } e \notin L_1 ; \\ I_1 \cup I_2, \text{ jestliže } e \in L_1 . \end{cases}$$

Přechodovou funkci δ definujme tímto předpisem:

(i) $\delta(q, a) = \delta_1(q, a)$,

jestliže

$q \in Q_1 \ \& \ \delta_1(q, a) \cap F_1 = \emptyset$,

(ii) $\delta(q, a) = \delta_1(q, a) \cup I_2$,

jestliže

$q \in Q_1 \;\&\; \delta_1(q, a) \cap F_1 \neq \emptyset$,

(iii) $\delta(q, a) = \delta_2(q, a)$,

jestliže

$q \in Q_2$.

Ověříme, že

$$L(\mathscr{A}) = L_1 \,.\, L_2 \,.$$

a) Nechť $w \in L_1 \,.\, L_2$. Potom slovo w je možné psát ve tvaru $w = uv$, kde $u \in L_1$, $v \in L_2$. Předpokládejme nejprve, že $u \neq e$, tzn. $u = a_1 \ldots a_n$ pro nějaká $a_1, \ldots, a_n \in \Sigma$, $n \geqq 1$. Potom existují stavy $s_1 \in I_1$ a $q \in Q_1$ takové, že slovo $a_1 \ldots a_{n-1}$ může převést automat $\mathscr{A}_1$ ze stavu s_1 do stavu q a $\delta_1(q, a_n) \cap F_1 \neq \emptyset$. Protože dále $v \in L_2$, existuje $s_2 \in I_2$ takový, že v může převést A_2 z s_2 do některého stavu v F_2. Podle definice δ tedy existuje výpočet automatu $\mathscr{A}$ začínající v s_1, který na základě $a_1 \ldots a_{n-1}$ přejde do q [používá se pravidlo typu (i) z definice δ], z tohoto stavu může $\mathscr{A}$ na základě a_n přejít do s_2 [pravidlo typu (ii)] a z s_2 na základě v do některého stavu z $F_2 = F$. To znamená, že $w = u \,.\, v \in L(\mathscr{A})$. Zbývá probrat případ $u = e$. V tom případě podle definice I může $\mathscr{A}$ začít výpočet přímo v s_2, a to výpočet, kterým $\mathscr{A}_2$ přechází do koncového stavu. Tedy opět

$$w = u \,.\, v \in L(\mathscr{A}) \,.$$

Dokázali jsme, že

$$L_1 \,.\, L_2 \subseteq L(\mathscr{A}) \,.$$

b) Předpokládejme, že $w \in L(\mathscr{A})$. Existuje tedy výpočet $\mathscr{A}$, kterým na základě w přejde z nějakého $s \in I$ do nějakého $r \in F$. Rozeberme dva případy.

$s \in I_2$: Potom podle definice I je $e \in L_1$. Kromě toho může automat $\mathscr{A}$ v tomto případě užívat pouze pravidla typu (iii)

z definice δ, tzn. sledovat přesně jistý výpočet automatu $\mathscr{A}_2$. Proto $w \in L_2$. Je tedy možné psát $w = e \,.\, w$, a protože $e \in L_1$ a $w \in L_2$, je $w \in L_1 \,.\, L_2$.

$s \in I_1$: Automat $\mathscr{A}$ tentokrát začíná svůj výpočet ve stavu z Q_1 a končí výpočet ve stavu patřícím do Q_2. Přechod z Q_1 do Q_2 může vykonat pouze pravidlem typu (ii) z definice δ. Je proto možné w psát ve tvaru $w = zav$, kde z je slovo odpovídající části výpočtu převádějící $\mathscr{A}$ z s do nějakého $q \in Q_1$ takového, že z q přejde na základě a do nějakého $s_2 \in I_2$, z něhož automat $\mathscr{A}$ přejde na základě v do F. Protože

$$\delta(q, a) \cap I_2 \neq \emptyset ,$$

musí podle definice δ být

$$\delta_1(q, a) \cap F_1 \neq \emptyset .$$

Z toho všeho plyne, že $za \in L_1$ a $v \in L_2$. Tedy

$$w = zav \in L_1 \,.\, L_2 .$$

Dokázali jsme, že také

$$L(\mathscr{A}) \subseteq L_1 \,.\, L_2 .$$

Platí tedy:

$$L(\mathscr{A}) = L_1 \,.\, L_2 .$$

2. Sestrojíme nedeterministický konečný automat

$$\tilde{\mathscr{A}} = (\tilde{Q}, \Sigma, \tilde{\delta}, \tilde{I}, \tilde{F})$$

takový, že

$$L(\tilde{\mathscr{A}}) = L_1^* .$$

Položme

$$\tilde{Q} = Q_1 \cup \{r\}$$

pro nějaké

$$r \notin Q_1 , \; I = I_1 \cup \{r\} , \; F = F_1 \cup \{r\} .$$

Přechodovou funkci $\bar{\delta}$ definujme takto:

(i) $\bar{\delta}(q, a) = \delta_1(q, a)$,

jestliže

$q \in Q_1$ a $\delta_1(q, a) \cap F_1 = \emptyset$,

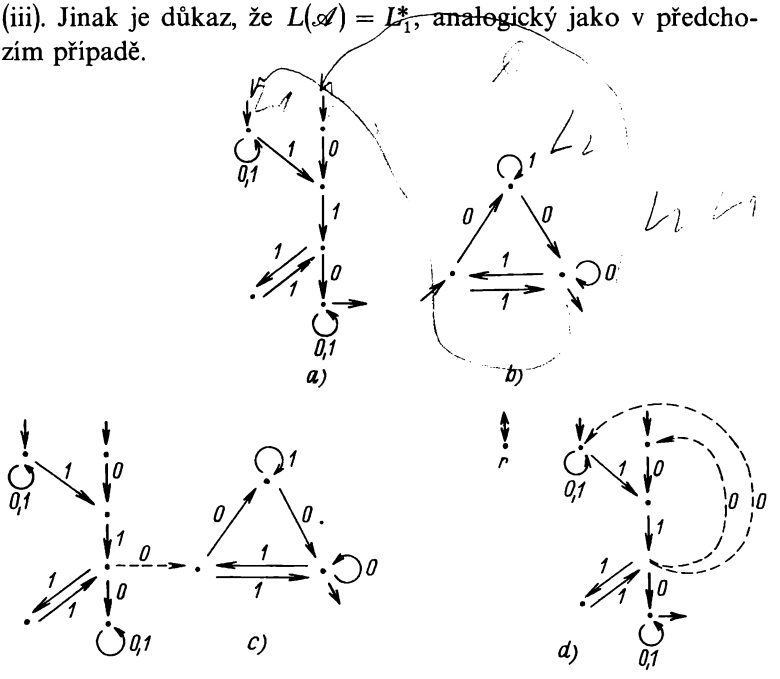
(ii) $\bar{\delta}(q, a) = \delta_1(q, a) \cup I_1$,

jestliže

$q \in Q_1$ a $\delta_1(q, a) \cap F_1 \neq \emptyset$,

(iii) $\delta(r, a) = \emptyset$ pro všechna $a \in \Sigma$.

Stav r ležící v $I \cap F$ zajišťuje, aby jazyk $L(\mathscr{A})$ obsahoval prázdné slovo. Žádné jiné slovo není z r možné přijmout díky pravidlům (iii). Jinak je důkaz, že $L(\mathscr{A}) = L_1^*$, analogický jako v předchozím případě.



Obr. 26

Příklad 2.2.16. K automatům $\mathscr{A}_1$, $\mathscr{A}_2$ z obr. 26a) a 26b) jsou na obr. 26c) a 26d) sestrojeny automaty $\mathscr{A}$ a $\tilde{\mathscr{A}}$ takové, že

$$L(\mathscr{A}) = L(\mathscr{A}_1) . L(\mathscr{A}_2) \text{ a } L(\tilde{\mathscr{A}}) = [L(\mathscr{A}_1)]^* .$$

(Nově přidané hrany jsou označeny čárkovaně.)

Ekvivalence nedeterministických konečných automatů s deterministickými využijeme také při důkazu uzavřenosti $\mathscr{F}$ vůči zrcadlovému obrazu.

Věta 2.2.17. *Pro libovolný jazyk L platí, že $L \in \mathscr{F}$, právě když $L^R \in \mathscr{F}$.*

Důkaz. Stačí dokázat, že

$$L \in \mathscr{F} \Rightarrow L^R \in \mathscr{F} ,$$

protože

$$(L^R)^R = L .$$

Z uvedené implikace pak už totiž dostaneme

$$L^R \in \mathscr{F} \Rightarrow (L^R)^R = L \in \mathscr{F} .$$

Nechť L je rozpoznáván nějakým nedeterministickým automatem

$$\mathscr{A} = (Q, \Sigma, \delta, I, F) .$$

Na základě $\mathscr{A}$ sestrojíme nedeterministický automat

$$\mathscr{B} = (Q, \Sigma, \delta', I', F')$$

rozpoznávající L^R. Názorně řečeno, stavový diagram nedeterministického automatu $\mathscr{B}$ dostaneme tak, že v diagramu $\mathscr{A}$ obrátíme orientaci hran, koncové stavy $\mathscr{A}$ uvažujeme jako počáteční stavy $\mathscr{B}$ a počáteční stavy $\mathscr{A}$ jako koncové stavy $\mathscr{B}$. Přesněji:

$$I' = F , \; F' = I$$

a

$$\bar{q} \in \delta'(q, a) \Leftrightarrow q \in \delta(\bar{q}, a)$$

pro všechna

$$q, \bar{q} \in Q \text{ a } a \in \Sigma .$$

Potom

a) $e \in L(\mathscr{A}) \Leftrightarrow I \cap F \neq \emptyset \Leftrightarrow$

$$\Leftrightarrow I' \cap F' \neq \emptyset \Leftrightarrow e \in L(\mathscr{B}) .$$

b) $a_1 \dots a_n \in L(\mathscr{A}) \Leftrightarrow$ existují $q_0, \dots, q_n$ takové, že

$q_0 \in I \;\&\; q_n \in F \;\&$

$\&$ [pro všechna i, $0 \leqq i \leqq n-1$ je $q_{i+1} \in \delta(q_i, a_{i+1})$] $\Leftrightarrow$

$\Leftrightarrow$ existují $q_n, \dots, q_0 \in Q$ takové, že $q_n \in I' \;\&\; q_0 \in F' \;\&$

$\&$ [pro všechna i, $n-1 \geqq i \geqq 0$ je $q_i \in \delta'(q_{i+1}, a_{i+1})$] $\Leftrightarrow$

$\Leftrightarrow a_n \dots a_1 \in L(\mathscr{B})$.

Tedy skutečně

$$L(\mathscr{B}) = L(\mathscr{A})^{\mathrm{R}} .$$

Jako poslední v tomto článku uvedeme důkaz uzavřenosti $\mathscr{F}$ vůči levému i pravému kvocientu.

Věta 2.2.18. *Jestliže* $L_1, L_2 \in \mathscr{F}$, *potom také* $L_2 \backslash L_1 \in \mathscr{F}$ *a* $L_1 / L_2 \in \mathscr{F}$.

Důkaz. 1. Nechť L_1 je rozpoznáván nedeterministickým konečným automatem

$$\mathscr{A}_1 = (Q_1, \Sigma, \delta_1, I_1, F_1)$$

a L_2 nedeterministickým konečným automatem

$$\mathscr{A}_2 = (Q_2, \Sigma, \delta_2, I_2, F_2) .$$

Zkonstruujeme nedeterministický konečný automat

$$\mathscr{A} = (Q, \Sigma, \delta, I, F)$$

takový, že

$$L(\mathscr{A}) = L_2 \backslash L_1 .$$

Nejprve sestrojme množinu stavů I automatu $\mathscr{A}_1$, do kterých je možné se dostat z některého stavu množiny I_1 na základě nějakého slova z jazyka L_2, tzn.

$$I = \{q \in Q_1 ; (\exists s \in I_1)(\exists u \in L_2)[q \in \delta(s, u)]\} .$$

Tuto množinu je možné zkonstruovat např. takto: pro každé $q \in Q_1$ definujme automat

$$\mathscr{A}_q = (Q_1, \Sigma, \delta_1, I_1, \{q\}) .$$

Potom zřejmě platí, že

$$q \in I \Leftrightarrow L(\mathscr{A}_q) \cap L_2 \neq \emptyset .$$

Podmínku

$$L(\mathscr{A}_q) \cap L_2 \neq \emptyset$$

můžeme efektivně testovat (viz větu 2.2.14). Množina I je množinou počátečních stavů automatu $\mathscr{A}$, jeho zbývající parametry definujme takto:

$$Q = Q_1 , \ \delta = \delta_1 , \ F = F_1 .$$

Ověřme, že

$$L(\mathscr{A}) = L_2 \setminus L_1 .$$

Pro libovolné slovo $v \in L(\mathscr{A})$ existuje výpočet vedoucí na základě v z nějakého stavu $s \in I$ do nějakého stavu $r \in F = F_1$. Stav s leží v I, a to podle definice I znamená, že existuje $u \in L_2$, které může převést automat $\mathscr{A}_1$ z nějakého stavu $s_1 \in I_1$ do s. Podle definice $\mathscr{A}$ pak ale uv převádí $\mathscr{A}_1$ z s_1 do r, a tedy $uv \in L_1$. Protože $uv \in L_1$ a $u \in L_2$, je $v \in L_2 \setminus L_1$.

Obráceně, ke každému slovu $v \in L_2 \setminus L_1$ existuje $u \in L_2$ tak, že $uv \in L_1$. Existuje proto výpočet $\mathscr{A}_1$ vedoucí na základě uv z nějakého $s_1 \in I_1$ do nějakého $r \in F_1$. Počáteční část tohoto výpočtu odpovídající slovu u převede $\mathscr{A}_1$ z s_1 do nějakého stavu s. Protože $u \in L_2$, je $s \in I$, a tedy $v \in L(\mathscr{A})$.

2. Uzavřenost $\mathscr{F}$ vůči pravému kvocientu plyne snadno z první části tohoto důkazu a uzavřenosti vůči zrcadlovému obrazu (viz 2.2.17), neboť

$$L_1/L_2 = ((L_2)^R \backslash (L_1)^R)^R .$$

Třída $\mathscr{F}$ je uzavřena i vůči substituci. Při důkazu tohoto tvrzení s výhodou využijeme regulárních výrazů, a proto se jím budeme zabývat až v následujícím článku.

2.3. Regulární jazyky, regulární výrazy

Formalismus, který zde zavedeme, je dalším užitečným nástrojem usnadňujícím návrh konečných automatů.

Definice 2.3.1. *Třída* $\mathrm{RJ}(\Sigma)$ *regulárních jazyků nad abecedou* Σ je nejmenší třída jazyků nad Σ splňující tyto podmínky:

1. $\emptyset \in \mathrm{RJ}(\Sigma)$ a $\{a\} \in \mathrm{RJ}(\Sigma)$ pro každé $a \in \Sigma$.
2. $L_1, L_2 \in \mathrm{RJ}(\Sigma) \Rightarrow L_1 \cup L_2 \in \mathrm{RJ}(\Sigma)$.
3. $L_1, L_2 \in \mathrm{RJ}(\Sigma) \Rightarrow L_1 . L_2 \in \mathrm{RJ}(\Sigma)$.
4. $L \in \mathrm{RJ}(\Sigma) \Rightarrow L^* \in \mathrm{RJ}(\Sigma)$.

Operace sjednocení, násobení a iterace se někdy nazývají *regulární operace nad jazyky*. Lze tedy také říci, že regulární jazyky jsou právě jazyky, které lze dostat z elementárních jazyků aplikací konečně mnoha regulárních operací. Elementárními jazyky zde rozumíme prázdný jazyk a jazyky tvořené jediným slovem o jednom symbolu.

Jak je z definice iterace 2.2.2 patrné, je i jazyk obsahující pouze prázdné slovo regulární, neboť $\{e\} = \emptyset^*$.

Každý regulární jazyk je zřejmě možné zadat stanovením příslušných elementárních jazyků a předpisu, jak na ně aplikovat regulární operace. K tomu slouží tzv. regulární výrazy.

Definice 2.3.2. *Množinu* $\mathrm{RV}(\Sigma)$ *regulárních výrazů nad abecedou*

$$\Sigma = \{a_1, \ldots, a_n\}$$

definujeme jako nejmenší množinu slov v abecedě

$$\{a_1, \ldots, a_n, \emptyset, e, +, ., *, (,)\}$$

[kde $e, \emptyset, +, ., *, (,)$ jsou symboly nepatřící do Σ] splňující tyto podmínky:

1. $\emptyset \in \mathrm{RV}(\Sigma)$,

   $e \in \mathrm{RV}(\Sigma)$,

   $a \in \mathrm{RV}(\Sigma)$ pro každé $a \in \Sigma$.

2. $\alpha, \beta \in \mathrm{RV}(\Sigma) \Rightarrow (\alpha + \beta) \in \mathrm{RV}(\Sigma), (\alpha . \beta) \in \mathrm{RV}(\Sigma)$,

   $\alpha^* \in \mathrm{RV}(\Sigma)$.

Každý z regulárních výrazů označuje jistý regulární jazyk. Výraz $\emptyset$ označuje prázdný jazyk, e označuje jazyk $\{e\}$, pro $a \in \Sigma$ označuje a jazyk $\{a\}$. Jestliže α, β jsou výrazy označující po řadě jazyky L_1, L_2, potom $(\alpha + \beta)$ označuje $L_1 \cup L_2$, $(\alpha . \beta)$ označuje $L_1 . L_2$ a α^* označuje L_1^*. Obecně budeme jazyk reprezentovaný regulárním výrazem α označovat $[\alpha]$.

Pro zpřehlednění zápisu regulárního výrazu zpravidla vynecháváme některé zbytečné závorky. Vynecháváme vnější pár závorek, dále můžeme vypustit některé závorky díky tomu, že operace součinu i sjednocení jazyků jsou operace asociativní. Proto můžeme např. místo $(((a . b) . c) . d)$ psát $a . b . c . d$ (nebo po vynechání teček $abcd$) a např. místo $((a + b) + (c + d))$ můžeme psát $a + b + c + d$. Další závorky můžeme vynechat na základě konvence o prioritě regulárních operací. Nejvyšší prioritu bude mít operace *, nejnižší operace +. Budeme tedy např. psát $a + (bc + d)^* a$ místo $(a + (((b . c) + d)^* . a))$.

Příklad 2.3.3. Regulární výraz

$$(01)^* 1111(01)^* + (0 + 1)^* 000$$

reprezentuje jazyk tvořený slovy, která buď obsahují podslovo 1111 předcházené i následované libovolným počtem dvojic 01, nebo slovy končícími na 000.

Věta 2.3.4. *Každý regulární jazyk je rozpoznatelný konečným automatem.*

Důkaz. Pro libovolnou konečnou abecedu Σ se snadno sestrojí automaty rozpoznávající jazyky $\emptyset$ a $\{a\}$ pro každé $a \in \Sigma$. Věta je tedy přímým důsledkem vět 2.2.11 a 2.2.15 o uzavřenosti $\mathscr{F}$ vůči regulárním operacím.

Poznámka 2.3.5. Důkazy vět 2.3.4, 2.2.11 a 2.2.15 naznačují, že každý regulární výraz α lze převést na nedeterministický automat $\mathscr{A}$ rozpoznávající $[\alpha]$ a přitom takový, že počet stavů automatu $\mathscr{A}$ nebude příliš převyšovat počet symbolů obsažených v α. Velikost $\mathscr{A}$ v závislosti na α můžeme odhadnout poměrně přesně: Každý z elementárních jazyků $\{a\}$ $(a \in \Sigma)$ lze reprezentovat nedeterministickým automatem o nejvýše dvou stavech, jazyky $\{e\}$ a $\emptyset$ automaty s jedním stavem. Důkazy vět 2.2.11 a 2.2.15 pak ukazují, že pokud regulárním výrazům α, β odpovídají po řadě nedeterministické automaty o m a n stavech, lze sestrojit nedeterministické automaty reprezentující $[\alpha + \beta]$ a $[\alpha\beta]$, z nichž každý má $m + n$ stavů. Automat reprezentující $[\alpha^*]$ bude mít $m + 1$ stavů.

Zvláštní význam dodává regulárním výrazům skutečnost, že jimi lze zadat každý jazyk rozpoznatelný konečným automatem.

Věta 2.3.6. *Každý jazyk rozpoznatelný konečným automatem je regulární.*

Důkaz. Buď $\mathscr{A} = (Q, \Sigma, \delta, q_1, F)$ konečný automat rozpoznávající L. Nechť

$$Q = \{q_1, \ldots, q_n\}.$$

Pro každé i, j $(1 \leqq i, j \leqq n)$ definujme

$$R_{ij} = \{w \in \Sigma^*; \delta(q_i, w) = q_j\},$$

tj. jako množinu slov, která převádějí automat ze stavu q_i do stavu q_j. Zřejmě platí, že

$$(*) \qquad L(\mathscr{A}) = L = \bigcup_{q_i \in F} R_{1i}.$$

Stačí proto dokázat, že každé R_{ij} je regulární jazyk. Protože L vzniká podle (*) konečným sjednocením (tj. konečným počtem regulárních operací) z jistých R_{1j}, plyne z regulárnosti R_{1j}, že i L je regulární.

Dokazujeme tedy, že pro každé i, j je R_{ij} regulární.

Definujme R_{ij}^k jako množinu slov převádějících automat ze stavu q_i do stavu q_j bez meziprůchodu jakýmkoliv stavem q_m takovým, že $m > k$. Pro každé i, j je zřejmě $R_{ij} = R_{ij}^n$, a proto stačí dokázat, že R_{ij}^k je regulární pro všechna i, j, k $(1 \leqq i, j, k \leqq n)$. Důkaz tohoto tvrzení provedeme indukcí podle k.

Základ indukce: Pro libovolná i, j je R_{ij}^0 množina slov, která převedou automat ze stavu q_i do q_j bez meziprůchodů jakýmkoli stavem. Taková slova, pokud vůbec existují, mají délku nejvýše 1. Je tedy

$$R_{ij}^0 \subseteq \Sigma \cup \{e\},$$

a proto je pro libovolná i, j množina R_{ij}^0 regulární.

Indukční krok: Předpokládejme, že pro jisté $0 \leqq k < n$ a všechna i, j je R_{ij}^k regulární. Dokážeme, že R_{ij}^{k+1} je regulární pro libovolná i, j. R_{ij}^{k+1} totiž můžeme vyjádřit ve tvaru

$$(**) \qquad R_{ij}^{k+1} = R_{ij}^k \cup R_{i,k+1}^k \cdot (R_{k+1,k+1}^k)^* \cdot R_{k+1,j}^k,$$

neboť jestliže w převádí automat z q_i do q_j bez meziprůchodu stavem s indexem vyšším než $k + 1$, mohou nastat tyto možnosti:

a) Nedojde ani k jednomu meziprůchodu stavem q_{k+1}. Potom $w \in R_{ij}^k$.

b) Dojde k m meziprůchodům stavem q_{k+1} $(m \geqq 1)$. Potom lze slovo w vyjádřit ve tvaru $w = uv_1v_2 \ldots v_{m-1}z$, kde m dělicích bodů mezi slovy $u, v_1, v_2, \ldots, v_{m-1}, z$ odpovídá jednotlivým meziprůchodům stavem q_{k+1}. Potom je

$$u \in R_{i,k+1}^k, \; z \in R_{k+1,j}^k \text{ a } v_l \in R_{k+1,k+1}^k$$

pro všechna $l \leqq m - 1$.
Proto je

$$w \in R_{i,k+1}^k \cdot (R_{k+1,k+1}^k)^* \cdot R_{k+1,j}^k.$$

Množina na pravé straně rovnosti (**) je tvořena čtyřmi regulárními operacemi z jazyků, které jsou podle indukčního předpokladu regulární. Je tedy také R_{ij}^{k+1} regulární pro všechna i, j. Tím je důkaz dokončen.

Shrnutím obou předchozích vět dostáváme výsledek známý jako Kleeneova věta.

Věta 2.3.7 (Kleene). *Libovolný jazyk je regulární, právě když je rozpoznatelný konečným automatem.*

Díky Kleeneově větě můžeme jednoduchým způsobem dokázat uzavřenost třídy $\mathscr{F}$ regulárních jazyků vůči substituci.

Věta 2.3.8. *Budiž Σ konečná abeceda a σ substituce taková, že pro každé $a \in \Sigma$ je $\sigma(a)$ regulární jazyk. Potom pro každý regulární jazyk L je $\sigma(L)$ regulární.*

Důkaz. Nechť α, α_a $(a \in \Sigma)$ jsou regulární výrazy takové, že α reprezentuje L a pro každé $a \in \Sigma$ výraz α_a reprezentuje $\sigma(a)$. Jestliže pro každé $a \in \Sigma$ dosadíme do α za každý výskyt symbolu $a \in \Sigma$ regulární výraz α_a, dostaneme regulární výraz reprezentující $\sigma(L)$.

Příklad 2.3.9. Nechť

$$\Sigma = \{0, 1\}, \ \sigma(0) = [(a + bc^*)^* bab],$$
$$\sigma(1) = [c + (dd)^*] \text{ a } L = [(011)^* (01 + e)].$$

Potom

$$\sigma(L) = [(((a + bc^*)^* bab)(c + (dd)^*)(c + (dd)^*))^* (((a + bc^*)^* bab)(c + (dd)^*) + e)].$$

Regulární výrazy určují jazyky specifikací vlastností jejich slov a díky tomu často umožňují zadávat jazyky rychle a přirozeně.

Příklad 2.3.10. Představme si situaci, kdy máme navrhnout automat sledující činnost jistého zařízení, o němž dostává informaci ve formě sledu nul a jedniček. Úkolem automatu je zjistit okamžik, kdy se v posloupnosti objeví jedna z kombinací 01001, 1111, 00101, z nichž každá signalizuje poruchu zařízení. Je vidět, že stačí sestrojit automat, který rozpoznává jazyk tvořený slovy končícími některou z uvedených kombinací. Takový automat se

dostane do jednoho z koncových stavů pokaždé, když zpracuje poslední symbol kterékoliv ze skupin charakterizujících poruchu. Popisovaný jazyk tedy stačí specifikovat regulárním výrazem

$$(0 + 1)^*(01001 + 1111 + 00101)$$

a od této specifikace lze na základě postupů, s kterými jsme se seznámili, přejít zcela mechanicky k nedeterministickému či deterministickému automatu a od něho k příslušné technické realizaci.

S úlohami podobného charakteru se můžeme setkat např. i při návrhu překladačů programovacích jazyků, když je třeba sestrojit lexikální analyzátor vyhledávající při průchodu zdrojovým programem vyhrazená slova (jako např. begin, while atd.), identifikátory apod.

Poznámka 2.3.11. Podle věty 2.2.11 jsou průnik dvou regulárních jazyků a doplněk regulárního jazyka opět regulární jazyky. K vytváření regulárních jazyků je tedy kromě operací sjednocování, řetězení a iterace možno přibrat navíc operaci průniku a operaci doplňku. V souhlase s tím se regulární výrazy obohatí o výrazy typu $\alpha \,\&\, \beta$ značící průnik jazyků reprezentovaných výrazy α a β a o výrazy typu $\bar{\alpha}$ označující doplněk jazyka reprezentovaného výrazem α. Mluvíme potom o *rozšířených regulárních výrazech*. Takové rozšíření někdy podstatně zjednodušuje zápis jazyka. Například jazyk v abecedě $\{0, 1\}$ tvořený právě všemi slovy obsahujícími alespoň jeden symbol 1 se zapíše výrazem $\overline{0^*}$, nebo jazyk v téže abecedě tvořený právě všemi slovy obsahujícími sudý počet symbolů 1 a sudý počet symbolů 0 lze zapsat výrazem

$$(0^*10^*10^*)^* \,\&\, (1^*01^*01^*)^* .$$

Zkuste pro zajímavost zapsat oba jazyky obyčejnými regulárními výrazy. Podobně můžeme rozšiřovat regulární výrazy na základě dalších operací, vůči nimž je třída $\mathscr{F}$ uzavřena.

U rozšířených regulárních výrazů ovšem ztrácíme možnost rychlého přechodu k nedeterministickému konečnému automatu, o které jsme mluvili v poznámce 2.3.5.

2.4. Regulární rovnice

K dosud probraným možnostem zadání regulárních jazyků připojíme ještě vymezení jazyka jakožto řešení soustavy regulárních rovnic.

Uveďme nejprve jednoduchý příklad.

Příklad 2.4.1. Nechť L je jazyk, jehož slova jsou tvořena sudým počtem jedniček následovaných skupinou 01110. Tuto charakteristiku můžeme vyjádřit rovnicí

$$(*) \qquad X = 11X + 01110\,.$$

Řešením této rovnice je takový regulární výraz α, že po jeho dosazení za X dostaneme ekvivalenci regulárních výrazů

$$\alpha \equiv 11\alpha + 01110\,, \text{ tzn. } [\alpha] = [11\alpha + 01110]\,.$$

Snadno se přesvědčíme, že takovým řešením je výraz

$$X = (11)^* 01110\,.$$

Obecně pro rovnici

$$(**) \qquad X = \alpha X + \beta\,,$$

kde α, β jsou regulární výrazy, je řešením např. každý regulární výraz

$$X = \alpha^*(\beta + \gamma)\,,$$

jakmile $e \in [\alpha]$. Většinou se zajímáme o minimální řešení, tj. výraz, který je řešením a přitom popisuje mezi všemi řešeními (**) nejmenší jazyk. Minimální řešení (**) je dáno jednoznačně a je rovno $X = \alpha^*\beta$. To plyne z následujícího tvrzení.

Věta 2.4.2. *Nechť α, β jsou regulární výrazy. Potom*

1. *Pro každý jazyk L vyhovující vztahu*

$$(***) \qquad L = [\alpha]\,.\,L \cup [\beta]$$

platí, že $L \supseteq [\alpha^*\beta]$.

2. *Jazyk $[\alpha^*\beta]$ vyhovuje vztahu (***), tzn. $\alpha^*\beta \equiv \alpha\alpha^*\beta + \beta$.*

Důkaz. 1. Ze vztahu (***) okamžitě plyne, že $[\beta] \subseteq L$ a jestliže $L \subseteq L$, potom $[\alpha].L \subseteq L$. Odtud se indukcí odvodí, že $[\alpha^*\beta] \subseteq L$.

2. Úpravou $\alpha\alpha^*\beta + \beta$ dostáváme

$$\alpha\alpha^*\beta + \beta \equiv \alpha^+\beta + \beta \equiv (\alpha^+ + e)\beta \equiv \alpha^* . \beta .$$

Tohoto poznatku se využívá při řešení tzv. standardních soustav regulárních rovnic.

Definice 2.4.3. *Standardní soustavou regulárních rovnic* o neznámých $X_1, \ldots, X_n$ nazýváme soustavu tvaru

$$X_i = \alpha_{i0} + \alpha_{i1}X_1 + \alpha_{i2}X_2 + \ldots + \alpha_{in}X_n, \quad 1 \leqq i \leqq n,$$

kde každé α_{ij} je regulární výraz nad nějakou abecedou disjunktní s $\{X_1, \ldots, X_n\}$.

Standardní soustavy regulárních rovnic se řeší podobně jako soustavy lineárních rovnic. Přitom využíváme znalosti řešení rovnice (**). Výsledkem je minimální řešení dané soustavy. Postup řešení si ukážeme na příkladu.

Příklad 2.4.4. Řešme soustavu

(1) $X_1 = (01^* + 1)X_1 + X_2$,

(2) $X_2 = 11 + 1X_1 + 00X_3$,

(3) $X_3 = e + X_1 + X_2$.

Výraz pro X_3 z (3) dosadíme do (2). Dostaneme soustavu

(4) $X_1 = (01^* + 1)X_1 + X_2$,

(5) $X_2 = 11 + 1X_1 + 00(e + X_1 + X_2) =$

$= 00 + 11 + (1 + 00)X_1 + 00X_2$.

Ze (4) vyjádříme X_1:

(6) $X_1 = (01^* + 1)^* X_2 = (0 + 1)^* X_2$.

[Úpravou podle zřejmého vztahu $(01^* + 1)^* \equiv (0 + 1)^*$.]

Dosazením do (5) získáme

$$(7) \qquad X_2 = 00 + 11 + (1 + 00)(0 + 1)^* X_2 + 00X_2 =$$
$$= 00 + 11 + (1 + 00)(0 + 1)^* X_2$$

(protože $00 \in [(1 + 00)(0 + 1)^*]$).

Ze (7) vyjádříme X_2:

$$(8) \qquad X_2 = ((1 + 00)(0 + 1)^*)^* (00 + 11).$$

Po dosazení do (6) máme

$$X_1 = (0 + 1)^* ((1 + 00)(0 + 1)^*)^* (00 + 11) =$$
$$= (0 + 1)^* (00 + 11).$$

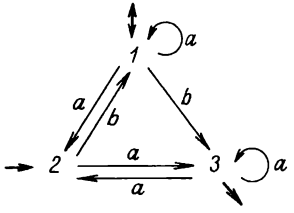
Konečně dosazením do (3) dostaneme

$$X_3 = e + (0 + 1)^* (00 + 11) +$$
$$+ ((1 + 00)(0 + 1)^*)^* (00 + 11) =$$
$$= e + (0 + 1)^* (00 + 11).$$

Popsaná metoda nabízí další způsob převodu konečných automatů (i nedeterministických) na regulární výrazy.

Příklad 2.4.5. Máme-li sestrojit regulární výraz pro jazyk rozpoznávaný automatem z obr. 27, pak pro všechna $i \in \{1, 2, 3\}$ označme proměnnou X_i výraz pro jazyk tvořený všemi slovy vedoucími ze stavu i do některého koncového stavu. Přímo podle stavového diagramu automatu sestavíme rovnice

$$X_1 = e + aX_1 + aX_2 + bX_3,$$
$$X_2 = bX_1 + aX_3,$$
$$X_3 = e + aX_2 + aX_3.$$

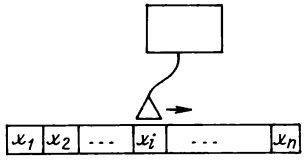


Obr. 27

Řešíme-li rovnice popsanou metodou, dostaneme pro proměnné X_1, X_2, X_3 po řadě jisté regulární výrazy α_1, α_2, α_3. Jazyk rozpoznávaný automatem je pak reprezentován výrazem $\alpha_1 + \alpha_2$ (poněvadž 1 a 2 jsou počáteční stavy).

2.5. Dvousměrné konečné automaty

Definice konečného automatu vychází z představy zařízení, které postupně zvnějšku dostává symboly vstupního řetězu. Takovýmto způsobem je vstupní slovo zpracováváno např. u zařízení s jednosměrnou vstupní páskou, tzn. páskou, na níž je zapsáno vstupní slovo a jejíž obsah je postupně čten hlavou, která se v každém taktu posouvá o jeden symbol doprava (viz obr. 28). Rozšíříme-li schopnosti konečného automatu o možnost posouvat hlavu po pásce v obou směrech, dostáváme tzv. dvousměrný konečný automat.



Obr. 28

Definice 2.5.1. *Dvousměrný konečný automat* nad vstupní abecedou Σ je pětice $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$, kde Q je konečná neprázdná množina stavů, Σ konečná neprázdná množina symbolů, $q_0 \in Q$ počáteční stav, $F \subseteq Q$ množina koncových stavů a $\delta: Q \times \Sigma \to Q \times \{-1, 0, +1\}$ je přechodová funkce.

Automat $\mathscr{A}$ pracuje nad libovolným slovem $w \in \Sigma^*$ tímto způsobem: Má-li na pásce napsáno slovo w, začne výpočet ve stavu q_0 na nejlevějším symbolu slova w. Je-li v nějakém okamžiku ve stavu q, čte symbol a, a je-li $\delta(q, a) = (q', p)$, pak přejde do stavu q' a posune se o políčko doleva (resp. doprava, resp. neposune se), podle toho, je-li $p = -1$ (resp. $p = +1$, resp. $p = 0$).

Slovo w je přijato, právě když během výpočtu nad w automat opustí pravý konec slova w v některém z koncových stavů. $L(\mathscr{A})$ označuje množinu slov přijímaných automatem $\mathscr{A}$, tj. jazyk rozpoznávaný tímto automatem.

Dvousměrné konečné automaty jsou zobecněním konečných automatů, a proto jsou samozřejmě schopny rozpoznat každý regulární jazyk. Dokážeme i zajímavé a na první pohled překvapující tvrzení, že jiné jazyky než regulární tento typ automatů není schopen rozpoznávat.

Věta 2.5.2. *Každý jazyk rozpoznatelný dvousměrným konečným automatem je regulární.*

Podrobný důkaz této věty může čtenář najít v knize [20]. Zde uvedeme jen jeho hlavní myšlenky jako ilustraci dalšího možného použití Nerodovy věty 1.3.3.

Buď $\mathscr{A} = (Q, \Sigma, \delta, q_0, F)$ libovolný dvousměrný konečný automat. Probereme výpočet tohoto automatu nad nějakým řetězem uv. Uvažujme, jakým způsobem se ve výpočtu nad úsekem v projeví skutečnost, že je předcházen úsekem u. Přítomnost u se projeví dvojím způsobem:

a) Výpočet započatý na levém konci řetězu u v počátečním stavu q_0 dospěje do situace, kdy čtecí hlava poprvé opouští pravý konec řetězu u v jistém stavu q (ve kterém vstoupí na úsek v). Druhou možností je výpočet, při němž hlava nikdy neopustí pravý konec úseku u (tzn. automat se dostane do cyklu nad řetězem u, resp. opustí slovo u na jeho levém konci).

Pro výpočet nad úsekem v má význam pouze informace, že automat poprvé vstoupí na v ve stavu q, resp. vůbec se na v nedostane. Ostatní vlastnosti řetězu u, či výpočtu nad ním, jsou z tohoto hlediska nepodstatné.

b) Jestliže někdy během výpočtu opustí automat levý konec řetězu v v nějakém stavu q', je pro další výpočet nad v důležité, v jakém stavu se opět na úsek v vrátí, resp. informace, že se na v nevrátí.

Tyto informace lze pro každý řetěz u vyjádřit jistou funkcí

$$f_u\colon \{\bar{q}_0\} \cup Q \to Q \cup \{0\},$$

kde $\bar{q}_0$, 0 jsou libovolné prvky nepatřící do Q. Hodnota $f_u(\bar{q}_0)$ udává informaci popsanou sub a), hodnoty $f(q)$ (pro všechna $q \in Q$) udávají informaci zmíněnou sub b). Hodnota 0 odpovídá případu, kdy pravá hranice úseku u už není v dalším výpočtu

překročena. Funkce f_u je tedy definována takto: jestliže úsek výpočtu zahájený na pravém (resp. levém) kraji řetězu u ve stavu $q \in Q$ (resp. q_0) vede k opuštění u na pravém konci ve stavu q', pak položíme

$$f_u(q) = q' \quad [\text{resp. } f_u(\bar{q}_0) = q'] .$$

Ve všech ostatních případech (tj. automat $\mathscr{A}$ opustí levý konec u nebo vůbec neopustí u) položíme

$$f_u(q) = 0 \quad [\text{resp. } f_u(\bar{q}_0) = 0] .$$

Definujme nyní na Σ^* relaci $\sim$ předpisem

$$w_1 \sim w_2 \Leftrightarrow_{\text{df}} f_{w_1} = f_{w_2} .$$

Tato relace je zřejmě ekvivalence. Jelikož existuje jen konečně mnoho různých funkcí $\{\bar{q}_0\} \cup Q \to Q \cup \{0\}$, je $\sim$ konečného indexu. Navíc pro každé $w \in \Sigma^*$ plyne z $w_1 \sim w_2$, že také $w_1 w \sim w_2 w$. Jestliže totiž $f_{w_1} = f_{w_2}$, bude výpočet nad úsekem w řetězu $w_1 w$ probíhat stejně jako výpočet nad w v řetězu $w_2 w$. Ekvivalence $\sim$ je tedy pravou kongruencí. Konečně jazyk $L(\mathscr{A})$ je sjednocením jistých tříd rozkladu Σ^* podle $\sim$. Pro každé slovo $w \in \Sigma^*$ je totiž zřejmě

$$w \in L(\mathscr{A}) \Leftrightarrow f_w(\bar{q}_0) \in F .$$

Proto také

$$w_1 \sim w_2 \Rightarrow [w_1 \in L(\mathscr{A}) \Leftrightarrow w_2 \in L(\mathscr{A})] .$$

Podle Nerodovy věty 1.3.3 je $L(\mathscr{A})$ regulární.

Každý dvousměrný konečný automat je podle naznačeného důkazu možné nahradit ekvivalentním jednosměrným konečným automatem, který může mít tolik různých stavů, kolik je různých funkcí

$$f: \{\bar{q}_0\} \cup Q \to Q \cup \{0\} ,$$

tzn. až $(n+1)^{(n+1)}$ stavů, jestliže $|Q| = n$. Podrobněji se k otázce rozdílu velikosti výchozího a výsledného automatu zmíněné transformace vrátíme v čl. 2.6.

V situaci, kdy čtecí hlava opustí vstupní slovo, není další krok výpočtu definován a výpočet končí. Při návrhu dvousměrného automatu nám proto bude často svazovat ruce možnost, že při posunu hlavy může dojít k neplánovanému překročení hranice slova. Vnucuje se otázka, zda není možné rozšířit možnosti dvousměrného automatu ohraničením vstupního slova koncovými znaky, které ho včas upozorní na nebezpečí opuštění pásky.

Definice 2.5.3. Řekneme, že *dvousměrný konečný automat* $\mathscr{A}$ *rozpoznává jazyk* $L = T(\mathscr{A})$ *s koncovými znaky*, jestliže pracovní abeceda automatu je $\Sigma \cup \{\#\}$ pro nějaký symbol $\# \notin \Sigma$ a

$$T(\mathscr{A}) = \{w \in \Sigma^*;\ \# w \# \in L(\mathscr{A})\}.$$

Díky poznatkům o uzávěrových vlastnostech regulárních jazyků můžeme snadno zodpovědět otázku, zda koncové znaky rozšiřují možnosti dvousměrných automatů.

Věta 2.5.4. *Každý jazyk rozpoznatelný dvousměrným konečným automatem s koncovými znaky je regulární.*

Důkaz. Budiž $T(\mathscr{A}) = \{w;\ \# w \# \in L(\mathscr{A})\}$ jako v definici 2.5.3. Potom

$$(*) \qquad T(\mathscr{A}) = \partial_{\#}\partial^{r}_{\#}(L(\mathscr{A}) \cap \# \Sigma^* \#).$$

$L(\mathscr{A})$ je podle 2.5.2 regulární, a protože regulární jazyky jsou uzavřeny vůči průniku (2.2.11) i vůči kvocientům, tzn. i derivacím, je podle $(*)$ i $T(\mathscr{A})$ regulární.

Díky větě 2.5.4 se např. podařilo zjistit některé překvapující uzávěrové vlastnosti regulárních jazyků, z nichž dvě uvedeme.

Důsledek 2.5.5. *Jestliže* L *je regulární jazyk, potom*

1. $L_1 = \{u;\ uu^{\mathrm{R}} \in L\}$ *a*
2. $L_2 = \{u;\ uu \in L\}$

jsou opět regulární jazyky.

Důkaz. Načrtneme důkaz tvrzení 2. Důkaz tvrzení 1 je zcela analogický. Jazyk L_2 (s koncovými znaky) lze rozpoznat dvousměrným konečným automatem $\mathscr{A}$ operujícím takto: $\mathscr{A}$ čte

vstupní slovo u zleva napravo a přitom simuluje činnost nějakého konečného automatu $\mathscr{A}_L$ rozpoznávajícího L. Jakmile narazí na konec slova, přeruší simulaci, znovu se vrátí na počátek u a pokračuje v simulaci $\mathscr{A}_L$. Jestliže v okamžiku, kdy dojde podruhé na konec slova, dospěje simulovaný výpočet $\mathscr{A}_L$ do koncového stavu, přijme $\mathscr{A}$ vstupní slovo.

Vedle důsledků věty 2.5.2 stojí za povšimnutí i její důkaz. Základní myšlenka tohoto důkazu se totiž v obecnější formě uplatňuje i v pokročilejších partiích teorie automatů při zkoumání různých typů automatů s dvousměrnou vstupní páskou. Podrobnější informace je možné nalézt např. v práci [8]. Jiný možný přístup budeme ilustrovat na důkazu zobecnění věty 2.5.4 pro případ nedeterministických automatů.

Definice 2.5.6. *Nedeterministický dvousměrný konečný automat* se vstupní abecedou je pětice $\mathscr{A} = (Q, \Sigma, \delta, I, F)$, kde Q je konečná množina stavů, Σ je konečná neprázdná vstupní abeceda, $I \subseteq Q$ je množina počátečních stavů, $F \subseteq Q$ je množina koncových stavů a

$$\delta: Q \times \Sigma \to \mathscr{P}(Q \times \{-1, 0, +1\})$$

je přechodová funkce.

Každý výpočet $\mathscr{A}$ nad libovolným slovem $w \in \Sigma^*$ začíná na prvním symbolu w v některém počátečním stavu. Je-li v nějakém okamžiku ve stavu q a čte symbol a, potom se může posunout o p políček doprava a přejít do stavu q', jestliže

$$(q', p) \in \delta(q, a).$$

Slovo w je přijato, jestliže některý z výpočtů nad ním končí opuštěním pravého konce slova v některém koncovém stavu. $L(\mathscr{A})$ označuje množinu slov přijímaných automatem $\mathscr{A}$, tj. jazyk rozpoznávaný tímto automatem.

Věta 2.5.7. *Každý jazyk rozpoznávaný nedeterministickým dvousměrným konečným automatem je regulární.*

Důkaz. Buď $\mathscr{A} = (Q, \Sigma, \delta, I, F)$ libovolný nedeterministický dvousměrný konečný automat. Každé slovo z $L(\mathscr{A})$ přijímá auto-

mat $\mathscr{A}$ nějakým výpočtem, při kterém žádné políčko není navštíveno podvakrát ve stejném stavu, protože část výpočtu mezi dvěma návštěvami políčka ve stejném stavu lze vynechat a vznikne opět přijímající výpočet. Výpočty uvedené vlastnosti nazývejme pro tuto chvíli minimální.

Naším cílem bude sestrojit nedeterministický konečný automat $\tilde{\mathscr{A}}$ přijímající právě slova, která jsou automatem $\mathscr{A}$ přijímána nějakým minimálním výpočtem. Z toho, co jsme právě ukázali, plyne, že $\tilde{\mathscr{A}}$ bude rozpoznávat jazyk $L(\mathscr{A})$. Tím bude podle věty 2.1.6 dokázáno, že $L(\mathscr{A})$ je regulární.

Přejděme ke konstrukci $\tilde{\mathscr{A}}$. Označme

$$\tilde{Q} = \mathscr{P}(Q) \times \mathscr{P}(Q).$$

Množina $\tilde{Q}$ je konečná a je stavovým prostorem automatu $\tilde{\mathscr{A}}$.

Vysvětleme si význam prvků množiny $\tilde{Q}$. Každý výpočet automatu $\mathscr{A}$ určuje pro každou hranici mezi dvěma políčky pásky tyto dvě množiny stavů:

1. množinu K_1 stavů, ve kterých automat přechází během výpočtu danou hranici směrem doprava,
2. množinu K_2 stavů, ve kterých přechází hranici směrem doleva.

U minimálních výpočtů dokonce každý prvek K_1 i K_2 odpovídá jedinému přechodu hranice. Prvky množiny $\tilde{Q}$ nazývejme *přechodovými dvojicemi.*

Pro libovolné přechodové dvojice (K_1, K_2), (K_1', K_2') a $a \in \Sigma$ budeme psát

$$(K_1, K_2) \xrightarrow{a} (K_1', K_2'),$$

jestliže existuje bijekce

$$h\colon K_1 \cup K_2' \to K_2 \cup K_1'$$

taková, že pro libovolný stav $q \in K_1 \cup K_2'$ platí toto:

a) Jestliže $h(q) \in K_2$, potom existují stavy

$$q_0, \ldots, q_k \quad (k \geqq 0)$$

takové, že

$$(q_{i+1}, 0) \in \delta(q_i, a)$$

pro všechna i, $0 \leqq i < k$, $q_0 = q$ a

$$(h(q), -1) \in \delta(q_k, a)$$

[tzn. ze stavu q na políčku se symbolem a lze přejít – popřípadě po konečném počtu taktů strávených na políčku – na levé sousední políčko ve stavu $h(q)$].

b) Jestliže $h(q) \in K_1'$, potom existují $q_0, \ldots, q_k$ $(k \geqq 0)$ taková, že

$$q_0 = q, (q_{i+1}, 0) \in \delta(q_i, a)$$

pro všechna i, $0 \leqq i < k$, a

$$(h(q), +1) \in \delta(q_k, a)$$

[tzn. ze stavu q na a lze první přechod uskutečnit doprava, a to ve stavu $h(q)$].

Každý přijímající výpočet nad libovolným vstupním slovem $a_1 \ldots a_n$ určuje pro hranice jednotlivých políček přechodové dvojice $\tilde{q}_0, \ldots, \tilde{q}_n \in \tilde{Q}$ tak, že

$$(*) \qquad \begin{cases} \tilde{q}_{i-1} \xrightarrow{a_i} \tilde{q}_i & \text{pro všechna } i,\ 1 \leqq i \leqq n, \\ \tilde{q}_n = (\{q\}, \emptyset) & \text{pro nějaké } q \in F, \\ \tilde{q}_0 = (\{q\}, \emptyset) & \text{pro nějaké } q \in I \end{cases}$$

(zde jsme se pro jednoduchost přidrželi představy, že na začátku výpočtu je hlava v některém počátečním stavu vložena na první políčko zleva).

Tím jsme ukázali, že každé slovo $w \in L(\mathscr{A})$ je přijímáno nedeterministickým konečným automatem

$$\tilde{\mathscr{A}} = (\tilde{Q}, \Sigma, \tilde{\delta}, \tilde{I}, \tilde{F}),$$

kde

$$\tilde{I} = \{(\{q\}, \emptyset); q \in I\}, \quad \tilde{F} = \{(\{q\}, \emptyset); q \in F\}$$

a

$$\tilde{p} \in \tilde{\delta}(\tilde{q}, a) \Leftrightarrow_{\text{df}} \tilde{q} \xrightarrow{a} \tilde{p}$$

pro libovolná $\tilde{p}, \tilde{q} \in \tilde{Q}$ a $a \in \Sigma$.

Zbývá dokázat, že $L(\tilde{\mathscr{A}}) \subseteq L(\mathscr{A})$. Nechť pro libovolné slovo $a_1 \dots a_n \in \Sigma^*$ existují stavy $\tilde{q}_0, \dots, \tilde{q}_n \in \tilde{Q}$ tak, že platí $(*)$. Ukažme, že $\tilde{q}_0, \dots, \tilde{q}_n$ určují nějaký minimální přijímající výpočet automatu $\mathscr{A}$ nad $a_1 \dots a_n$.

Jestliže $\tilde{q}_0 = (\{q\}, \emptyset)$ a $\tilde{q}_1 = (K_1^1, K_2^1)$, potom funkce h (jejíž existence je zajištěna podmínkou $\tilde{q}_0 \xrightarrow{a_1} \tilde{q}_1$) udává jistý stav $h(q) \in K_1^1$, v němž hlava může v souhlase s přechodovou funkcí δ přejít na symbol a_2. Stavu $h(q)$ je stejným způsobem přiřazen stav, v němž stroj může (doleva nebo doprava) opustit symbol a_2. Takto se postupně určí přípustný výpočet stroje, který podle $(*)$ nutně skončí opuštěním pravého konce slova v koncovém stavu. Tím jsme ověřili, že jazyk $L(\mathscr{A})$ je rozpoznáván automatem $\tilde{\mathscr{A}}$.

Obdobně jako v 2.5.4 se ukáže, že i každý jazyk rozpoznatelný nedeterministickým dvousměrným konečným automatem s koncovými znaky je regulární. Z toho pak analogicky jako v 2.5.5 plyne např. toto:

Důsledek 2.5.8. *Jestliže L je regulární jazyk, potom*

1. $L_1 = \{u;\ uv \in L,\ \text{pro nějaké } v \text{ takové, že } |v| = |u|\}$,
2. $L_2 = \{u;\ u = wv \text{ pro nějaká } w, v \text{ taková, že } w^R wv \in L\}$

jsou opět regulární jazyky.

2.6. Ekonomie popisu regulárních jazyků

U všech alternativních způsobů reprezentace jazyků rozpoznatelných konečnými automaty jsme popsali algoritmus přechodu k ekvivalentnímu konečnému automatu. Zatím jsme se příliš nestarali, jak velký tento automat bude, protože konec konců může být jen mezivýsledkem, ze kterého lze redukcí získat vůbec nejmenší možný konečný automat reprezentující daný jazyk.

Uvažujeme-li však o praktickém použití popsaných algoritmů, dojdeme k otázce, zda jsme jako mezivýsledek nepřipouštěli zbytečně velké automaty. Tak např. algoritmus přechodu od nedeterministického k deterministickému automatu 2.1.6 teoreticky připouští, aby z výchozího nedeterministického automatu o n stavech byl vytvořen deterministický automat o 2^n stavech. Zkon-

struovat takový „mezivýsledek" ovšem může být prakticky neproveditelné i při poměrně malých n. Například nedeterministický automat o třiceti stavech a dvouprvkové abecedě lze reprezentovat tabulkou, která se vejde na jednu stránku. Řádky tabulky deterministického automatu o 2^{30} stavech by vyžadovaly přes 35 miliónů stránek.

Proto je důležité prozkoumat možnost, že existuje algoritmus převodu nedeterministického automatu na deterministický, u kterého by tak prudký vzrůst velikosti stavového prostoru nenastával.

Podobnou otázku vyvolává důkaz věty 2.2.17 o uzavřenosti třídy regulárních jazyků vůči zrcadlovému obrazu. I zde náš algoritmus teoreticky připouští exponenciální vzrůst stavového prostoru. Nebylo by i zde možné postupovat úsporněji?

Odpověď na obě otázky dává následující věta.

Věta 2.6.1. *Ke každému $n > 0$ existuje regulární jazyk L_n v abecedě $\{a, b\}$ takový, že*

1. *L_n je rozpoznatelný nedeterministickým konečným automatem o n stavech,*
2. *redukovaný deterministický automat rozpoznávající L_n má 2^n stavů,*
3. *L_n^R je rozpoznatelný deterministickým automatem o $2n$ stavech.*

Důkaz. Budiž $n > 0$ libovolné, $\Sigma = \{a, b\}$. Jazyk L_n zadáme nedeterministickým automatem

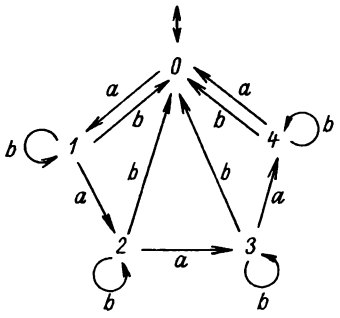
$$\mathscr{A}_n = (Q_n, \Sigma, \delta_n, I_n, F_n),$$

který je definován takto:

$$\begin{aligned}
&Q_n = \{0, 1, 2, \ldots, n-1\}, \quad I_n = F_n = \{0\}, \\
&\delta_n(i, a) = \{i+1\} \quad \text{pro všechna } i,\ 0 \leqq i < n-1, \\
&\delta_n(n-1, a) = \{0\}, \\
&\delta_n(i, b) = \{0, i\} \quad \text{pro všechna } i,\ 0 < i \leqq n-1, \\
&\delta_n(0, b) = \emptyset.
\end{aligned}$$

(Na obr. 29 je znázorněn automat $\mathscr{A}_5$.)

Pro každé $n > 0$ položme $L_n = L(\mathscr{A}_n)$. Ověřme, že pro tyto jazyky platí tvrzení 1 až 3 věty. Zvolme libovolné pevné $n > 0$. Tvrzení 1 plyne přímo z definice L_n.



Obr. 29

Ověřme tvrzení 2: Z $\mathscr{A}_n$ sestrojme podmnožinovou konstrukcí deterministický automat $\hat{\mathscr{A}}_n$ ekvivalentní s $\mathscr{A}_n$. Automat $\hat{\mathscr{A}}_n$ má 2^n stavů. Ukážeme, že $\hat{\mathscr{A}}_n$ je redukovaný, a tím bude tvrzení 2 dokázáno.

a) Všechny stavy $\hat{\mathscr{A}}_n$ jsou dosažitelné z počátečního stavu $K_0 = \{0\}$; speciálně stav $K = \emptyset$ je z K_0 dosažitelný symbolem b. Jestliže

$$\emptyset \neq K = \{i_1, \ldots, i_r\}, \quad 0 \leqq i_1 < \ldots < i_r \leqq n - 1,$$

potom slovo

$$a^{i_r - i_{r-1}} b a^{i_{r-1} - i_{r-2}} b \ldots a^{i_3 - i_2} b a^{i_2 - i_1} b a^{i_1}$$

převede automat $\hat{\mathscr{A}}_n$ ze stavu K_0 do K, jak lze snadno ověřit indukcí. [Např. slovo a^2ba^2b převede automat $\hat{\mathscr{A}}_5$ (viz obr. 29) do stavu $\{4, 2, 0\}$ a slovo a^2ba^2 převede tentýž automat do stavu $\{4, 2\}$.]

b) Žádné dva různé stavy automatu $\hat{\mathscr{A}}_n$ nejsou ekvivalentní: nechť K_1 a K_2 jsou libovolné dva různé stavy $\hat{\mathscr{A}}_n$. Bez újmy na obecnosti můžeme předpokládat, že existuje $i \in K_1 - K_2$. Symbol a určuje v $\mathscr{A}_n$ deterministicky přechod do následujícího stavu, tzn. pro libovolné i, j je

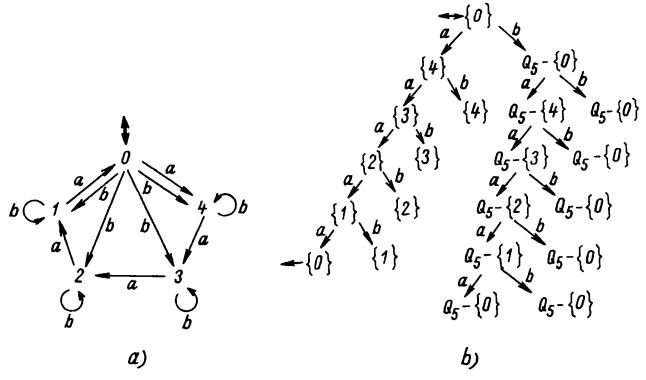
$$\delta(i, a^j) = \{k\},$$

kde

$$k = (i + j) \bmod n .$$

Proto slovo a^{n-i} převede $\hat{\mathscr{A}}_n$ ze stavu K_1 do koncového stavu, tj. stavu obsahujícího 0, zatímco totéž slovo převede $\hat{\mathscr{A}}_n$ ze stavu K_2 do stavu, který není koncový. Stavy K_1 a K_2 proto nejsou ekvivalentní.

Zbývá ověřit tvrzení 3. Podle 2.2.17 přejdeme od automatu $\mathscr{A}_n$ k automatu $\mathscr{B}_n$ rozpoznávajícímu L_n^R [na obr. 30a) je automat $\mathscr{B}_5$].



a) b)

Obr. 30

Při přechodu od $\mathscr{B}_n$ k ekvivalentnímu deterministickému automatu $\hat{\mathscr{B}}_n$ (podmnožinovou konstrukcí) dostáváme pouze $2n$ dosažitelných stavů:

$$\{0\}, \{1\}, \{2\}, \ldots, \{n-1\},$$

$$Q_n - \{0\}, Q_n - \{1\}, \ldots, Q_n - \{n-1\} .$$

[Na obr. 30b) je stavový diagram pro $\hat{\mathscr{B}}_5$.]

Tím je tvrzení 3 dokázáno.

Věta, kterou jsme právě dokázali, je malou, ale důležitou ukázkou obecnějšího jevu, který si zaslouží podrobnější komentář.

Poznámka 2.6.2 (O stručnějším popisu silnějšími prostředky). Přesvědčili jsme se, že zobecnění deterministických automatů na nedeterministické někdy přináší možnost výrazně úspornější re-

prezentace jazyka. Ve větě 2.6.4 ukážeme, že dvousměrné konečné automaty někdy dovolují ještě pronikavější zkrácení reprezentace jazyka. Podobný efekt přináší vyjádření jazyků jako výsledku operací nad jednoduššími jazyky, jak jsme se přesvědčili v tvrzení 3 předchozí věty nebo jak může čtenář sám ověřit, zkusí-li vyjádřit jazyky L_n z věty 2.6.1 pomocí regulárních výrazů.

V dalších kapitolách se seznámíme s prostředky pro popis širších tříd jazyků zahrnujících třídu regulárních jazyků jako podmnožinu. Silnější prostředky určené pro jejich reprezentaci poskytují zhuštěnější popisy některých regulárních jazyků. Již na úrovni prostředků schopných reprezentovat tzv. bezkontextové jazyky (kterými se budeme podrobněji zabývat v kap. 4) lze docílit neomezeného zkrácení popisu některých regulárních jazyků ve srovnání s popisem konečnými automaty.

Rozveďme podrobněji, v jakém smyslu zde mluvíme o neomezeném zkrácení popisu. Zvolíme-li libovolně rychle rostoucí funkci f, kterou lze zadat algoritmem, potom k ní existuje nekonečně mnoho n s touto vlastností: Existuje regulární jazyk L_n popsatelný bezkontextovou gramatikou délky n (či zásobníkovým automatem s přechodovou tabulkou rozsahu n apod.), ale reprezentovaný redukovaným konečným automatem, který má více než $f(n)$ stavů.

Poznámka 2.6.3 (O nevýhodách silnějších prostředků). Proč se zabýváme studiem konečných automatů, regulárních výrazů atp., jestliže jsou k dispozici prostředky umožňující reprezentovat jednak všechny regulární jazyky, a to mnohdy podstatně stručněji než zmíněné nástroje, jednak ještě další, neregulární jazyky?

Důvod je praktický. Sestrojení konečné reprezentace jazyka není zpravidla cílem samo o sobě, ale je východiskem pro další postup – konstrukci zařízení, zkoumání vlastností daného jazyka apod. V této další fázi nabízejí jednodušší popisovací formalismy výhody, které přechodem k silnějším prostředkům ztrácíme.

Víme například, jak od popisu konečným automatem nebo regulárním výrazem apod. přejít víceméně mechanicky k odpovídajícímu konečnému zařízení (sekvenčnímu obvodu, programu používajícímu konečný úsek paměti apod.). Taková reprezentace

dokonce dovoluje okamžitě odhadnout potřebný rozsah paměti – deterministický automat o n stavech bude při své realizaci vyžadovat maximálně $\log_2 n$ bitů paměti, nedeterministický automat o n stavech povede na realizaci s maximálně n bity paměti atd. Silnější prostředky, jako např. už zmíněné bezkontextové gramatiky, vedou na složitější systémy vyžadující potenciálně nekonečnou paměť.

Podobně se při přechodu k silnějším prostředkům komplikuje ověřování vlastností jazyka. Uveďme si jako příklad tyto vlastnosti jazyka (v jisté abecedě Σ):

a) jazyk je prázdný,

b) jazyk je roven Σ^*,

c) jazyk je roven jinému danému jazyku,

d) rozpoznávání jazyka lze realizovat sekvenčním obvodem.

Zabýváme-li se pouze jazyky reprezentovanými deterministickými konečnými automaty, není problém ověřit kteroukoliv z uvedených vlastností. Vlastnost d) je splněna triviálně; ověřit a) pouze znamená zjistit, zda některý koncový stav automatu je dosažitelný; b) se snadno ověří přechodem k doplňku jazyka a podle a); rychlý postup pro ověření c) jsme popsali v 1.5.23.

Přejdeme-li k reprezentaci jazyků pomocí nedeterministických konečných automatů, zůstává d) triviálním problémem a také a) lze snadno ověřit. Otázky b) a c) lze teoreticky vždy vyřešit přechodem k ekvivalentním deterministickým konečným automatům. To ovšem nemusí být prakticky proveditelný postup, jak vyplývá z věty 2.6.1. Dodnes není znám algoritmus, který by ve všech případech vedl dostatečně rychle k zodpovězení b) a c) a mnohé nasvědčuje tomu, že takový algoritmus neexistuje (kniha [2]). U prostředků schopných reprezentovat právě všechny bezkontextové jazyky se už otázky b), c), d) stávají algoritmicky nerozhodnutelné (viz kap. 6), zatímco a) lze poměrně snadno rozhodnout (viz 4.1.2).

Přechodem k ještě silnějším prostředkům (např. pro kontextové jazyky) se i a) stává algoritmicky nerozhodnutelnou otázkou.

Z tohoto hlediska je třeba pohlížet na různé typy popisu regulárních jazyků i na klasifikaci formálních jazyků, se kterou se seznámíme v kap. 3. Obojí je diktováno snahou získat teoretické

nástroje, které jsou „šité na míru" určitým důležitým třídám problémů.

Na závěr kapitoly uvedeme výsledek týkající se popisu regulárních jazyků dvousměrnými konečnými automaty. Algoritmus z důkazu věty 2.5.2 počítal s tím, že při přechodu od dvousměrného automatu s n stavy k ekvivalentnímu jednosměrnému automatu se může počet stavů zvýšit až na $(n+1)^{n+1}$. Ukážeme, že tuto horní hranici nelze podstatně snížit. (V důkazu věty si všimněte, že jazyky, na kterých je stručnější popis dvousměrnými automaty demonstrován, jsou dokonce konečné.)

Věta 2.6.4. *Ke každému* $n > 1$ *existuje konečný jazyk*

$$L_n \subseteq \{a, b, c, d\}^*$$

takový, že

1. *je rozpoznatelný dvousměrným konečným automatem (s koncovými znaky) s nejvýše* $5n + 5$ *stavy,*

2. *redukovaný konečný automat rozpoznávající* L_n *má alespoň* n^n *stavů.*

Důkaz. Jazyk L_n definujme předpisem

$$L_n = \{a^{i_1}ba^{i_2}b \ldots ba^{i_n}c^k d^{i_k} \; ; \; 1 \leqq k \leqq n$$

a $1 \leqq i_j \leqq n$ pro všechna j, $1 \leqq j \leqq k\}$.

1. Dvousměrný automat rozpoznávající L_n pracuje takto:

i) postupuje přes a a b, dokud nenarazí na c, pak

ii) se vrací až k levému koncovému znaku a přitom kontroluje, zda procházený úsek slova obsahuje právě $n - 1$ symbolů b,

iii) od levého koncového znaku postupuje doprava a kontroluje, zda žádná skupina stejných sousedních symbolů a není delší než n, potom

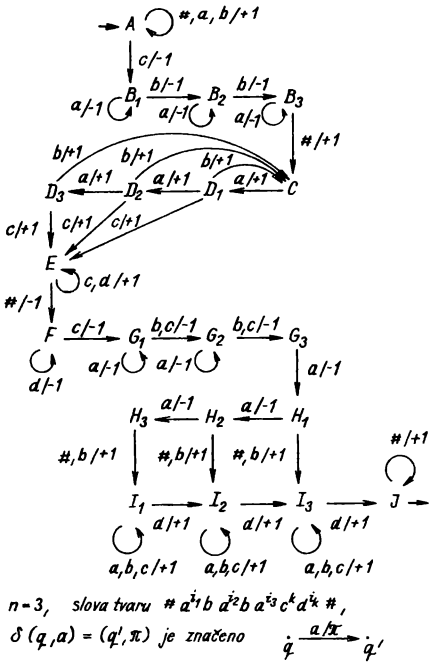
iv) zjistí, zda za prvním symbolem c jsou jenom symboly c a d, od koncového znaku se vrátí k c,

v) určí počet symbolů c, tzn. k a vrátí se o $n - k$ symbolů b doleva,

vi) změří příslušnou skupinu symbolů a, pak postupuje doprava a

vii) porovná ji se skupinou symbolů d,

viii) v úspěšném případě skončí přijetím slova.



Obr. 31

Čtenář se sám může přesvědčit, že k tomu stačí $5n + 5$ stavů. Na obr. 31 je graf takového automatu pro L_3. Jednotlivým částem výpočtu odpovídá značení stavů podle tabulky

i) A,

ii) B,

iii) D,

iv) E, F,

v) G,

vi) H,

vii) I,

viii) J.

2. Budiž $\mathscr{A}_n$ redukovaný automat rozpoznávající jazyk L_n. Podle 1.3.3 k $\mathscr{A}_n$ sestrojíme pravou kongruenci $\sim$ konečného indexu takovou, že sjednocení některých tříd ekvivalence dává L_n. Stačí ukázat, že tato kongruence má alespoň n^n tříd:

Jazyk L_n má n^n různých prefixů tvaru

$$a^{i_1}ba^{i_2}b \dots ba^{i_n}.$$

Dokážeme, že různé prefixy padnou do různých tříd ekvivalence, a tím bude důkaz uzavřen.

Budiž

$$a^{i_1}ba^{i_2} \dots ba^{i_n} \neq a^{j_1}ba^{j_2} \dots ba^{j_n}.$$

Potom $i_k \neq j_k$ pro nějaké k, $1 \leqq k \leqq n$. Kdyby byly oba prefixy kongruentní, platilo by

$$a^{i_1}b \dots ba^{i_n}c^k d^{i_k} \sim a^{j_1}b \dots ba^{j_n}c^k d^{i_k}.$$

Přitom první slovo patří do L_n a druhé nikoliv. To je spor s předpokladem, že L_n je sjednocením jistých tříd ekvivalence. Žádné dva z uvedených prefixů tedy nejsou kongruentní, a proto tyto prefixy padnou do n^n různých tříd ekvivalence. Tedy $\mathscr{A}_n$ má alespoň n^n stavů.

Z předchozích kapitol známe dva důležité a navzájem odlišné typy konečné reprezentace jazyků. První způsob, reprezentace pomocí automatů, poskytuje mechanismus, který umožňuje o každém slovu rozhodnout, zda do daného jazyka patří, či nikoliv. Druhý způsob, charakterizace regulárními výrazy, určuje jazyk tak, že specifikuje vlastnosti slov, která do něho patří.

Tato kapitola obohatí náš repertoár o další typ reprezentace jazyků, totiž o reprezentaci pomocí gramatik, tzn. soustav pravidel, podle kterých se vytvářejí právě všechny řetězy daného jazyka.

Aparát formálních gramatik byl původně navržen pro popis přirozených jazyků. Novou rozsáhlou oblast použití získal jako nástroj pro zadávání vyšších programovacích jazyků. Od dob zavedení Algolu 60 je už zcela běžné, že definice určující, co jsou syntakticky správně napsané programy v určitém jazyce, mívají formu gramatiky.

Gramatiky spolu s automaty jsou nejčastější způsoby reprezentace jazyků a vztahy mezi nimi budou tvořit rámec následujících tří kapitol. V nich nám půjde zejména o to, jakým způsobem přejít od pravidel určujících správné tvoření slov jazyka, tj. od gramatiky, k zařízení rozpoznávajícímu příslušnost slov k jazyku, či dokonce provádějícímu gramatický rozbor slov. Budeme se zabývat i metodami, jak obráceně na základě automatu určujícího, která slova do jazyka patří, najít pro příslušný jazyk gramatiku.

ZÁKLADNÍ ČÁST

3.1. Přepisovací systémy, gramatiky

Vytváření slov podle určité gramatiky se děje postupným přepisováním řetězů v souhlase s jejími pravidly.

Definice 3.1.1. *Přepisovací systém* (*produkční systém*) je uspořádaná dvojice $\mathscr{R} = (V, P)$, kde V je konečná abeceda a P je konečná množina přepisovacích pravidel. Každé *přepisovací pravidlo* (*produkce*) je uspořádaná dvojice (u, v), kde $u, v \in V^*$. Přepisovací pravidlo (u, v) zpravidla zapisujeme $u \to v$.

Definice 3.1.2. Nechť $\mathscr{R} = (V, P)$ je přepisovací systém, $u, z \in V^*$.

1. Řekneme, že u *se přímo přepíše na* z, symbolicky $u \Rightarrow_{\mathscr{R}} z$ (nebo prostě $u \Rightarrow z$, pokud je z kontextu zřejmé, že se jedná o systém $\mathscr{R}$), právě když existují slova $v, w, x, y \in V^*$ taková, že $u = vxw$, $z = vyw$ a $x \to y$ je přepisovací pravidlo z P.

2. Řekneme, že slovo u *se přepíše na* z, symbolicky $u \Rightarrow^*_{\mathscr{R}} z$ (nebo $u \Rightarrow^* z$), právě když existuje posloupnost

$$(*) \qquad u_1, u_2, \ldots, u_n \ (n \geqq 1)$$

slov z V^* takových, že

$$u = u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_n = z\,.$$

Posloupnost $(*)$ nazýváme *odvozením* (nebo *derivací*) *délky n slova z ze slova u*. Odvození $(*)$ se nazývá *minimální*, jestliže $u_i \neq u_j$ pro všechna $i \neq j$. Místo „u se přepíše na z" říkáme také „z u se odvodí z", „z u dostaneme z", „u generuje z" atp. Všimněte si, že relace $\Rightarrow^*$ je reflexívní a tranzitivní. (Je to reflexívní a tranzitivní uzávěr relace $\Rightarrow$.)

Příklad 3.1.3. Nechť

$$\mathscr{R} = (V, P)$$

je přepisovací systém, kde

$$V = \{0, 1\}\,, \quad P = \{01 \to 10, 10 \to 01\}\,.$$

Potom např.

$$00110 \Rightarrow^* 00011\,,$$

neboť

$$00\underline{11}0 \Rightarrow 00\underline{10}1 \Rightarrow 00011$$

(výskyt levé části použitého pravidla je pro větší přehlednost podtržen). Ale je také

$$00110 \Rightarrow^* 01100,$$

neboť

$$\underline{001}100 \Rightarrow 01\underline{010} \Rightarrow 01100.$$

Je na první pohled vidět, že tento přepisovací systém může přepsat každé slovo $u \in \{0,1\}^*$ na libovolné jiné slovo $v \in \{0,1\}^*$ se stejným počtem výskytů symbolů 0 a 1 jako má u.

Lemma 3.1.4. *Jestliže $\mathscr{R}$ je přepisovací systém a $u \Rightarrow^* z$ pro u, $z \in V^*$, potom existuje minimální odvození slova z z u.*

Důkaz plyne přímo z definice. Jestliže totiž derivace není minimální, obsahuje úsek tvaru $u_i \Rightarrow \ldots \Rightarrow u_j$, kde $u_i = u_j$. Celý tento úsek je možno vypustit a nahradit slovem u_i. Po konečném počtu takových kroků dostaneme minimální odvození.

Kdykoliv tedy budeme v dalším textu mluvit o derivaci, můžeme bez újmy na obecnosti předpokládat, že se jedná o minimální derivaci.

Definice 3.1.5. *Generativní gramatika* je uspořádaná čtveřice $\mathscr{G} = (\Pi, \Sigma, S, P)$, kde Π a Σ jsou dvě disjunktní konečné abecedy, $S \in \Pi$, a P je konečná množina přepisovacích pravidel tvaru $\alpha \to \beta$, kde α, β jsou slova z abecedy $\Pi \cup \Sigma$, přičemž α obsahuje alespoň jeden symbol z abecedy Π.

Π se nazývá *množina neterminálů* (*proměnných*), Σ se nazývá *množina terminálů* a S je *počáteční symbol.*

Generativní gramatika $\mathscr{G} = (\Pi, \Sigma, S, P)$ určuje přepisovací systém $(\Pi \cup \Sigma, P)$. Výrazy „(přímo) generuje", „odvození" atd. v souvislosti s gramatikou $\mathscr{G}$ podobně jako symboly $\Rightarrow$, $\Rightarrow^*$ chápeme jako tyto výrazy a symboly definované pro přepisovací systém $(\Pi \cup \Sigma, P)$.

Jazyk $L(\mathscr{G})$ generovaný generativní gramatikou $\mathscr{G}$ je definován takto:

$$L(\mathscr{G}) = \{w;\ w \in \Sigma^* \text{ a } S \Rightarrow_{\mathscr{G}}^* w\}.$$

Je tedy tvořen všemi slovy v terminální abecedě, která lze odvodit z počátečního symbolu.

Příklad 3.1.6. 1. Nechť

$$\mathcal{G} = (\Pi, \Sigma, S, P)$$

je generativní gramatika, kde

$$\Pi = \{S\}\,,\ \Sigma = \{0, 1\}\,,\ P = \{S \to 0S1, S \to 01\}\,.$$

Okamžitě je vidět, že

$$L(\mathcal{G}) = \{0^n 1^n;\ n \geqq 1\}\,.$$

2. Nechť

$$\mathcal{G} = (\Pi, \Sigma, S, P)\,,$$

kde

$$\Pi = \{S, B, C\}\,,\ \Sigma = \{a, b, c\}\,.$$

P nechť je tvořeno těmito pravidly:

(1) $S \to SaBC$,

(2) $S \to aBC$,

(3) $aB \to Ba$,

(4) $Ba \to aB$,

(5) $aC \to Ca$,

(6) $Ca \to aC$,

(7) $BC \to CB$,

(8) $CB \to BC$,

(9) $B \to b$,

(10) $C \to c$.

V tomto případě $L(\mathcal{G})$ obsahuje právě všechna neprázdná slova ze Σ^*, v nichž se vyskytuje stejný počet symbolů a, b i c. Na ukázku odvoďme slovo *caabbcbac*:

$$S \Rightarrow_{(1)} SaBC \Rightarrow_{(1)} SaBCaBC \Rightarrow_{(2)} aBCaBCaBC \Rightarrow_{(7)}$$
$$\Rightarrow_{(7)} aCBaBCaBC \Rightarrow_{(5)} CaBaBCaBC \Rightarrow_{(10)}$$

$$\Rightarrow_{(10)} caBaBCaBC \Rightarrow_{(4)} caaBBCaBC \Rightarrow_{(9)}$$

$$\Rightarrow_{(9)} caabBCaBC \Rightarrow_{(9)} caabbCaBC \Rightarrow_{(10)}$$

$$\Rightarrow_{(10)} caabbcaBC \Rightarrow_{(3)} caabbcBaC \Rightarrow_{(10)}$$

$$\Rightarrow_{(10)} caabbcBac \Rightarrow_{(9)} caabbcbac .$$

V odvození jsou pro lepší orientaci zapsána i čísla použitých pravidel.

Ověřme, že gramatika skutečně generuje daný jazyk. Nejprve ukažme, že každé slovo z $L(\mathcal{G})$ obsahuje stejný počet symbolů a, b, c. Nechť $w \in L(\mathcal{G})$. Potom $S \Rightarrow^* w$ a existuje odvození

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \ldots \Rightarrow w_n = w \quad \text{pro nějaké } n \geqq 1 .$$

Z tvaru přepisovacích pravidel této gramatiky plyne, že všechna slova v odvození mají nutně tuto vlastnost:

Jestliže $u(x)$ označuje počet symbolů x ve slově u, potom

$$w_i(a) = w_i(b) + w_i(B) = w_i(c) + w_i(C), \quad 1 \leqq i \leqq n .$$

Protože $w_n = w$ obsahuje pouze terminály, je

$$w_n(B) = w_n(C) = 0 ,$$

a tedy

$$w(a) = w(b) = w(c) .$$

Zbývá ukázat, že každé neprázdné slovo tvořené stejným počtem symbolů a, b, c patří do $L(\mathcal{G})$.

Zvolme proto libovolné slovo $w \in \Sigma^*$ obsahující n ($n \geqq 1$) symbolů a, b, c. Zvolené slovo vytvoříme takto:

I. Nejprve $(n-1)$-krát použijeme pravidlo (1), potom jednou pravidlo (2). Tak vznikne slovo $(aBC)^n$.

II. Pomocí pravidel (2) až (8) přeskupíme symboly ve slově $(aBC)^n$ tak, aby vzniklo slovo, které se liší od slova w pouze tím, že na místě b stojí vždy B a na místě c stojí C.

III. Pomocí pravidel (9) a (10) změníme všechna B na b a všechna C na c.

Příklad 3.1.7. Uveďme gramatiku, která generuje právě všechny zápisy čísel přípustné v jazyce Algol 60. Gramatiku zapíšeme

v tzv. *Backusově–Naurově formě* (BNF). V ní se místo symbolu → užívá ::=, neterminály jsou značeny slovy uzavřenými do závorek ⟨ ⟩ a všechna pravidla se stejnou levou stranou se shrnou do jednoho zápisu tak, že se zapíše společná levá strana, potom znak ::= a napravo od něho postupně všechny pravé strany pravidel oddělené svislými čarami.

Pravidla gramatiky vypadají takto:

⟨číslo⟩ ::= ⟨číslo bez znaménka⟩ |
+ ⟨číslo bez znaménka⟩ |
− ⟨číslo bez znaménka⟩,

⟨číslo bez znaménka⟩ ::= ⟨desetinné číslo⟩ |
⟨exponentová část⟩ |
⟨desetinné číslo⟩ ⟨exponent. část⟩,

⟨desetinné číslo⟩ ::= ⟨celé číslo bez znaménka⟩ |
⟨desetinná část⟩ |
⟨celé číslo bez znaménka⟩ ⟨desetinná část⟩,

⟨exponentová část⟩ ::= $_{10}$⟨celé číslo⟩,

⟨desetinná část⟩ ::= .⟨celé číslo bez znaménka⟩,

⟨celé číslo⟩ ::= ⟨celé číslo bez znaménka⟩ |
+ ⟨celé číslo bez znaménka⟩ |
− ⟨celé číslo bez znaménka⟩,

⟨celé číslo bez znaménka⟩ ::= ⟨číslice⟩ |
⟨celé číslo bez znaménka⟩ ⟨číslice⟩,

⟨číslice⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9.

Počátečním symbolem této gramatiky je neterminál ⟨číslo⟩, terminály jsou 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, −, ., $_{10}$, celkem 14 terminálních symbolů. Zkuste sami podle této gramatiky vygenerovat slova 299, $-.981_{10}-06$ nebo .1234, která jsou přípustnými zápisy čísel v Algolu 60.

Předchozí příklad názorně ilustroval generativní gramatiku jakožto návod, podle kterého se vytvářejí všechna slova z jazyka reprezentovaného gramatikou. Jsme-li postaveni před poněkud odlišnou úlohu prokázat příslušnost daného slova k jazyku zadanému gramatikou, snažíme se najít odvození tohoto slova z počátečního symbolu. Toto odvození je někdy jednodušší konstruovat pozpátku. Vyjdeme od zadaného slova a jeho jednotlivé skupiny symbolů spojujeme do větších a větších celků, které nahrazujeme příslušnými neterminály, až se nakonec v souladu s gramatickými pravidly podaří celé výchozí slovo redukovat na počáteční symbol.

Na tento postup jsme zvyklí např. z rozboru vět živého jazyka, jak jsme jej prováděli ve škole, nebo z rozboru cizího programu v programovacím jazyku.

Tak např. při rozboru textu, o kterém zjišťujeme, zda je českou větou, postupně usuzujeme takto:

Toto je podstatné jméno, toto je přídavné jméno, toto podstatné a přídavné jméno tvoří podmět atd., až nakonec můžeme usoudit, že předložený řetěz symbolů je gramaticky správná česká věta.

Podobně při rozboru programu můžeme uvažovat takto: ... toto je identifikátor, toto znak přiřazení, toto je číslo, dohromady tvoří přiřazovací příkaz, ... to je podmíněný příkaz, ... toto je blok, ... je to program v Algolu 60.

Popsaný přístup vede k definici pojmu duálního k pojmu generativní gramatiky.

Definice 3.1.8. *Analytická gramatika* je uspořádaná čtveřice $\mathscr{G} = (\Pi, \Sigma, S, P)$, kde Π, Σ, S jsou stejné jako v definici generativní gramatiky a P je konečná množina uspořádaných dvojic $\alpha \to \beta$ takových, že $\alpha, \beta \in (\Pi \cup \Sigma)^*$ a β obsahuje alespoň jeden neterminální symbol.

Analytická gramatika opět určuje přepisovací systém $(\Pi \cup \Sigma, P)$, pomocí něhož lze zavést relace $\Rightarrow_{\mathscr{G}}$, $\Rightarrow^*_{\mathscr{G}}$. *Jazyk* $L(\mathscr{G})$ *rozpoznávaný analytickou gramatikou* $\mathscr{G}$ je definován takto:

$$L(\mathscr{G}) = \{w;\ w \in \Sigma^* \text{ a } w \Rightarrow^* S\}.$$

Je zřejmé, že „obrácením šipek" v pravidlech libovolné generativní gramatiky $\mathscr{G}$ dostaneme analytickou gramatiku $\mathscr{G}'$ takovou, že $L(\mathscr{G}) = L(\mathscr{G}')$, a obráceně, stejným způsobem lze od analytické gramatiky přejít k ekvivalentní generativní gramatice. Z toho je vidět, že oba pojmy jsou rovnocenné a že se můžeme omezit buď pouze na generativní, nebo pouze na analytické gramatiky. V dalším výkladu se proto budeme zabývat pouze generativními gramatikami, takže slovo „generativní" budeme vypouštět a budeme mluvit prostě o gramatikách.

3.2. Chomského hierarchie

Některé jazyky lze generovat i gramatikami, které mají přepisovací pravidla speciálního typu, tj. nepoužívají všech možností nabízených obecnou definicí gramatiky. Má proto smysl třídit gramatiky podle tvarů pravidel, která obsahují. Nejstarší a nejznámější klasifikací gramatik založenou na tomto principu je klasifikace, kterou zavedl N. Chomsky.

Definice 3.2.1. 1. Gramatiku $\mathscr{G}$ v obecné formě tak, jak byla definována v 3.1.5, budeme nazývat *gramatikou typu* 0. Jazyk generovaný gramatikou typu 0 se nazývá *jazyk typu* 0.

2. Gramatika $\mathscr{G} = (\Pi, \Sigma, S, P)$ se nazývá *gramatika typu* 1 nebo *kontextová gramatika*, právě když každé přepisovací pravidlo z P je tvaru

$$\alpha X \beta \to \alpha \gamma \beta ,$$

kde $\alpha, \beta \in (\Pi \cup \Sigma)^*$, $X \in \Pi$ a $\gamma \in (\Pi \cup \Sigma)^+$ (tzn. $|\gamma| \geqq 1$). Jedinou výjimkou může být pravidlo $S \to e$, jehož výskyt však znamená, že S se nesmí objevit na pravé straně žádného přepisovacího pravidla z P.

Jazyk typu 1 nebo *kontextový jazyk* je každý jazyk, který lze generovat nějakou kontextovou gramatikou.

3. *Gramatika typu* 2 neboli *bezkontextová gramatika* je každá gramatika $\mathscr{G} = (\Pi, \Sigma, S, P)$, kde P obsahuje pouze pravidla typu

$$X \to \gamma ,$$

kde $X \in \Pi$ a $\gamma \in (\Pi \cup \Sigma)^*$.

Jazyk se nazývá *bezkontextový* nebo *typu* 2, právě když jej lze generovat nějakou bezkontextovou gramatikou.

4. *Gramatika* $\mathcal{G} = (\Pi, \Sigma, S, P)$ je *typu* 3 neboli *regulární* (nebo také *pravá lineární*), jestliže každé pravidlo z P je buď tvaru

$$X \to wY, \text{ nebo tvaru } X \to w,$$

kde $X, Y \in \Pi$, $w \in \Sigma^*$.

Jazyk se nazývá *regulární* neboli *typu* 3, jestliže jej lze generovat regulární gramatikou.

Poznámka 3.2.2. Název „regulární jazyk" jsme zavedli už v 2.3 pro jazyk reprezentovaný regulárním výrazem. V čl. 3.3 ukážeme, že se v obou případech jedná o tutéž třídu jazyků.

Poznámka 3.2.3. Použití kontextového pravidla $\alpha X \beta \to \alpha \gamma \beta$ i bezkontextového pravidla $X \to \gamma$ má stejný efekt: Neterminál X se přepíše na γ, ostatní části slova zůstanou nezměněny. V prvním případě však k tomu dojde jen tehdy, jestliže X je obklopeno řetězy α a β, tj. možnost přepsat X na γ závisí na kontextu, ve kterém se X vyskytuje. Ve druhém případě přepsání X na kontextu nezávisí. Toto pozorování vysvětluje, proč se gramatiky typu 1 nazývají kontextové a gramatiky typu 2 bezkontextové.

Poznámka 3.2.4. Poslední věta v definici kontextové gramatiky hovořící o pravidle $S \to e$ je do definice pojata pouze proto, aby další teorie byla co nejjednodušší, speciálně aby platilo tvrzení 3.2.6. Jiný význam tento požadavek nemá. Na druhé straně je podmínka, aby $|\gamma| \geqq 1$, velmi důležitá.

Označení 3.2.5. Symbolem $\mathcal{L}_i$ (pro $i = 0, 1, 2, 3$) budeme značit třídu všech jazyků typu i.

Tvrzení 3.2.6. $\mathcal{L}_0 \supseteq \mathcal{L}_1 \supseteq \mathcal{L}_2 \supseteq \mathcal{L}_3$.

Důkaz. Inkluze $\mathcal{L}_0 \supseteq \mathcal{L}_1$, $\mathcal{L}_2 \supseteq \mathcal{L}_3$ plynou přímo z definice 3.2.1, neboť každá gramatika typu 1 je také typu 0 a každá gramatika typu 3 je také typu 2. Takový vztah neplatí mezi gramatikami typu 2 a typu 1, protože bezkontextové gramatiky mohou např. obsahovat větší počet pravidel typu $X \to e$, zatímco

kontextová gramatika může obsahovat pouze jediné pravidlo s prázdným slovem na pravé straně, a to $S \to e$, přičemž S se pak nesmí vyskytovat na pravé straně žádného pravidla. Vztah $\mathscr{L}_1 \supseteq \mathscr{L}_2$ proto plyne až z věty 3.2.8, která ukazuje, že každou bezkontextovou gramatiku lze upravit tak, aby splňovala požadavky kladené na kontextové gramatiky a přitom stále generovala původní jazyk.

Tato věta nám kromě toho později poslouží jako užitečné technické lemma při studiu bezkontextových jazyků.

Definice 3.2.7. *Bezkontextová gramatika* se nazývá *nevypouštějící*, jestliže neobsahuje žádné přepisovací pravidlo tvaru $X \to e$.

Věta 3.2.8. *Ke každé bezkontextové gramatice $\mathscr{G}$ existuje nevypouštějící bezkontextová gramatika $\mathscr{G}_1$ taková, že*

$$L(\mathscr{G}_1) = L(\mathscr{G}) - \{e\}.$$

Jestliže $e \in L(\mathscr{G})$, potom existuje bezkontextová gramatika

$$\mathscr{G}' = (\Pi', \Sigma, S', P')$$

taková, že $L(\mathscr{G}) = L(\mathscr{G}')$, jediné pravidlo s e na pravé straně je $S' \to e$ a S' se nevyskytuje na pravé straně žádného z pravidel v P'.

Důkaz. Ke gramatice $\mathscr{G} = (\Pi, \Sigma, S, P)$ zkonstruujeme gramatiku $\mathscr{G}_1 = (\Pi, \Sigma, S, P_1)$. Nejprve sestrojíme množinu

$$U = \{X \in \Pi;\ X \Rightarrow^* e\}.$$

Tuto množinu vytváříme postupně. Nejdříve do ní zahrneme všechny neterminály X takové, že P obsahuje pravidlo $X \to e$. Potom prohlížíme množinu P a přidáme do množiny U všechny neterminály X takové, že v P je pravidlo $X \to \alpha$ a $\alpha \in U^*$. S takto rozšířenou množinou U opakujeme stejnou proceduru a pak opět, tak dlouho, až do U není možno přidat nový neterminál. Ideu konstrukce můžeme matematicky přesně vyjádřit takto: definujeme

$$U_1 = \{X \in \Pi;\ X \to e \in P\},$$

$$\cdots\cdots\cdots$$

$$U_{i+1} = U_i \cup \{X;\ \text{existuje } \alpha \in U_i^* \text{ takové, že } X \to \alpha \in P\}.$$

Zřejmě platí, že $U_1 \subseteq U_2 \subseteq \ldots \subseteq \Pi$. Protože Π je konečná, existuje přirozené číslo n, pro které je $U_n = U_{n+1}$. Z definice $U_1, U_2, \ldots$ plyne, že $U_n = U_{n+k}$ pro každé $k \geq 1$. Proto $U_n = U$.

Na základě U sestrojíme množinu P_1 takto: P_1 obsahuje pravidlo $X \to \alpha$, právě když

1. $\alpha \neq e$ a současně
2. v P je obsaženo pravidlo

$$X \to \beta\,,\ \beta = \gamma_1 Y_1 \gamma_2 Y_2 \ldots \gamma_m Y_m \gamma_{m+1}\,,$$

kde $m \geq 0$,

a) všechna Y_i patří do U,

b) v žádném γ_i se už symbol z U nevyskytuje a

c) α lze dostat z β vynecháním některých (třeba žádných) symbolů z $Y_1, \ldots, Y_m$.

Dokažme, že

$$L(\mathcal{G}_1) = L(\mathcal{G}) - \{e\}\,.$$

Je celkem zřejmé, že

$$L(\mathcal{G}_1) \subseteq L(\mathcal{G}) - \{e\}\,.$$

Každé odvození podle $\mathcal{G}_1$ lze totiž simulovat odvozením podle $\mathcal{G}$ takto: jestliže v odvození podle $\mathcal{G}_1$ je použito některé z pravidel, která má oproti $\mathcal{G}$ navíc, např. pravidlo $X \to \alpha$, kde α vzniklo z $\beta = \gamma_1 Y_1 \ldots Y_m \gamma_{m+1}$ vypuštěním některých Y_i, potom na příslušném místě odvození podle $\mathcal{G}$ se použije pravidlo $X \to \beta$ a potom se z příslušných Y_i vygeneruje e, což je možné, neboť $Y_i \in U$, a tedy $Y_i \Rightarrow_{\mathcal{G}}^{*} e$. Tím jsme dokázali, že

$$L(\mathcal{G}_1) \subseteq L(\mathcal{G})\,.$$

Jelikož $\mathcal{G}_1$ neobsahuje podle definice P_1 žádné vypouštějící pravidlo, nemůže být v $L(\mathcal{G}_1)$ obsaženo prázdné slovo, a proto

$$L(\mathcal{G}_1) \subseteq L(\mathcal{G}) - \{e\}\,.$$

Ukažme ještě, že

$$L(\mathcal{G}) - \{e\} \subseteq L(\mathcal{G}_1)\,.$$

Nechť

$$w \in L(\mathscr{G}) - \{e\}.$$

Jestliže při odvození $S \Rightarrow_{\mathscr{G}}^{*} w$ byla použita pouze pravidla z $P_1 \cap P$, je také $w \in L(\mathscr{G})$. Zbývá probrat případ, kdy byla použita pravidla z P, která nepatří do P_1. To jsou pouze pravidla tvaru $X \to e$. Jejich použití lze v $\mathscr{G}_1$ obejít vhodným použitím pravidel, která má $\mathscr{G}_1$ navíc oproti $\mathscr{G}$. Jestliže totiž je v odvození w podle $\mathscr{G}$ použito pravidlo $X \to e$, pak $Y \Rightarrow^{*} X \Rightarrow e$ pro nějaký neterminál Y, který byl v příslušném odvození vygenerován nějakým pravidlem $Z \to \eta Y \xi$, kde η nebo ξ se dále přepíší na neprázdné slovo. Toto použití pravidla můžeme nahradit pravidlem $Z \to \eta' \xi'$, kde η' a ξ' vzniknou z η a ξ vypuštěním těch výskytů neterminálů, které se v dalším přepíší na prázdné slovo. Pravidlo $Z \to \eta' \xi'$ je podle definice v P_1. Tím je dokázána první část věty.

Předpokládejme nyní, že $e \in L(\mathscr{G})$. Nejprve zkonstruujeme gramatiku $\mathscr{G}_1$ jako v první části důkazu a potom vytvoříme gramatiku

$$\mathscr{G}' = (\Pi \cup \{S'\}, \Sigma, S', P_1 \cup \{S' \to e, S' \to S\}).$$

Pro tuto gramatiku platí, že

$$L(\mathscr{G}') = L(\mathscr{G}_1) \cup \{e\} = L(\mathscr{G})$$

a množina pravidel má požadované vlastnosti.

V dalším textu budeme u bezkontextových gramatik pravidla se stejnou levou stranou zapisovat zkráceně tak, jak jsme se s tím setkali již v př. 3.1.7, tzn. např. místo

$$X \to \alpha_1,$$
$$X \to \alpha_2,$$
$$\ldots\ldots\ldots$$
$$X \to \alpha_n$$

budeme psát

$$X \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n.$$

Často budeme gramatiky zadávat pouze množinou jejich pravidel. V tom případě neterminály budou označeny velkými písme-

ny, ostatní znaky budou terminály. Počáteční symbol bude označen S. Na všechny odchylky od této úmluvy budeme výslovně upozorňovat.

Příklad 3.2.9. Gramatiku $\mathscr{G}$:

$$S \to aSbA \mid e,$$

$$A \to aBbA \mid bCb \mid CD,$$

$$B \to bbBa \mid aS,$$

$$C \to aAaA \mid e,$$

$$D \to SC \mid aABb$$

převedeme (postupem z 3.2.8) na nevypouštějící gramatiku $\mathscr{G}_1$ generující jazyk $L(\mathscr{G}) - \{e\}$

$$U_1 = \{S, C\},$$

$$U_2 = \{S, C, D\},$$

$$U_3 = \{S, C, D, A\},$$

$$U_4 = U_3 = U.$$

Gramatika $\mathscr{G}_1$ vypadá takto:

$$S \to aSbA \mid abA \mid aSb \mid ab,$$

$$A \to aBbA \mid aBb \mid bCb \mid bb \mid CD \mid D \mid C,$$

$$B \to bbBa \mid aS \mid a,$$

$$C \to aAaA \mid aaA \mid aAa \mid aa,$$

$$D \to SC \mid C \mid S \mid aABb \mid aBb.$$

Pro ilustraci uveďme odvození slova $aabbbb$ v gramatice $\mathscr{G}$ a jemu odpovídající odvození v $\mathscr{G}_1$:

$\mathscr{G}$: $S \Rightarrow aSbA \Rightarrow aaSbAbA \Rightarrow aabAbA \Rightarrow aabCDbA \Rightarrow$
$\Rightarrow aabDbA \Rightarrow aabSCbA \Rightarrow aabCbA \Rightarrow aabbA \Rightarrow$
$\Rightarrow aabbbCb \Rightarrow aabbbb,$

$\mathscr{G}_1$: $S \Rightarrow aSbA \Rightarrow aabbA \Rightarrow aabbbb.$

3.3. Regulární gramatiky a jazyky

V tomto článku ukážeme, že jazyky typu 3 jsou právě jazyky rozpoznatelné konečnými automaty a že je tedy název „regulární jazyky" zavedený v 3.2.1 v souladu s předchozím názvoslovím z čl. 2.3.

Věta 3.3.1. *Nechť L je jazyk rozpoznatelný konečným automatem. Potom L je jazyk typu 3.*

Důkaz. Nechť $L = L(\mathscr{A})$ pro nějaký konečný automat

$$\mathscr{A} = (Q, \Sigma, \delta, q_0, F).$$

Definujme gramatiku $\mathscr{G} = (Q, \Sigma, q_0, P)$ takto:

$$P = \{q \to aq'; \delta(q, a) = q', q, q' \in Q, a \in \Sigma\} \cup$$
$$\cup \{q \to e; q \in F\}.$$

Z definice $\mathscr{G}$ je vidět, že pro libovolné $a_1 \dots a_n \in \Sigma^+$ a stav $q \in Q$ je $q_0 \Rightarrow_{\mathscr{G}}^* a_1 \dots a_n q$, právě když $\mathscr{A}$ ze stavu q_0 přejde na základě $a_1 \dots a_n$ do q. Proto

$$a_1 \dots a_n \in L(\mathscr{A}) \Leftrightarrow q_0 \Rightarrow_{\mathscr{G}}^* a_1 \dots a_n q$$
$$\text{pro nějaké } q \in F$$
$$\Leftrightarrow q_0 \Rightarrow^* a_1 \dots a_n,$$

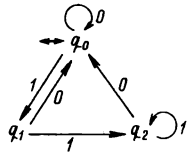
neboť $q \to e$ patří do P právě pro koncová q. Pro každé neprázdné slovo w je tedy

$$w \in L(\mathscr{A}) \Leftrightarrow w \in L(\mathscr{G}).$$

Probereme ještě zvlášť případ prázdného slova

$$e \in L(\mathscr{A}) \Leftrightarrow q_0 \in F \Leftrightarrow q_0 \to e \in P \Leftrightarrow e \in L(\mathscr{G}).$$

Úhrnem dostáváme, že $L(\mathscr{G}) = L(\mathscr{A})$.



Obr. 32

Příklad 3.3.2. Automatu z obr. 32 odpovídá regulární gramatika

$$q_0 \to 0q_0 \mid 1q_1 \mid e,$$

$$q_1 \to 0q_0 \mid 1q_2,$$

$$q_2 \to 0q_0 \mid 1q_2,$$

v níž q_0, q_1, q_2 jsou neterminály, 0, 1 terminály a q_0 je počáteční symbol.

K větě 3.3.1 platí také věta obrácená. Než to dokážeme, vyslovme jednoduché pomocné tvrzení.

Lemma 3.3.3. *Ke každé gramatice typu* 3 *existuje s ní ekvivalentní gramatika typu* 3, *která má pouze pravidla tvaru*

$$X \to aY, \quad X \to Y \quad \textit{nebo} \quad X \to e,$$

kde X, Y *jsou neterminály a* a *je terminál.*

Důkaz. Budiž $\mathscr{G} = (\Pi, \Sigma, S, P)$ libovolná gramatika typu 3. Sestrojme gramatiku $\mathscr{G}' = (\Pi, \Sigma, S, P')$, kde P' bude

(1) obsahovat všechna pravidla typu

$$X \to aY, \quad X \to Y, \quad X \to e$$

patřící do P,

(2) místo každého pravidla tvaru

$$X \to a_1 \dots a_n Y \quad (n \geqq 2, a_1, \dots, a_n \in \Sigma)$$

patřícího do P bude do P' zahrnuta soustava pravidel

$$X \to a_1 Y_2, \quad Y_2 \to a_2 Y_3, \dots, Y_n \to a_n Y,$$

kde $Y_2, \dots, Y_n$ jsou nově přidané neterminály nevyskytující se v žádných jiných pravidlech P',

(3) místo každého pravidla tvaru

$$Z \to b_1 \dots b_m \quad (m \geqq 1, b_1, \dots, b_m \in \Sigma)$$

patřícího do P bude do P' zahrnuta soustava pravidel

$$Z \to b_1 Z_1,\ Z_1 \to b_2 Z_2, \ldots, Z_{m-1} \to z_m Z_m,\ Z_m \to e,$$

kde $Z_1, \ldots, Z_m$ jsou opět nově přidané neterminály.

Množina Π' bude tvořena všemi neterminály vyskytujícími se v pravidlech P'.

Nově vytvořená gramatika je zřejmě požadovaného tvaru a je s $\mathscr{G}$ ekvivalentní, neboť aplikaci každého pravidla $\mathscr{G}$ nepatřícího do $\mathscr{G}'$ lze v $\mathscr{G}'$ nahradit postupnou aplikací odpovídající soustavy pravidel. Obráceně každé pravidlo $\mathscr{G}'$ nepatřící do $\mathscr{G}$ je částí některé soustavy popsané sub (2) a (3). Protože nově přidané neterminály se nevyskytují mimo příslušnou soustavu, je nutné v každém odvození terminálního slova podle $\mathscr{G}'$ takovou soustavu aplikovat celou v několika na sebe navazujících přepsáních. Aplikace celé soustavy pravidel je ovšem rovnocenná aplikaci odpovídajícího pravidla $\mathscr{G}$ a lze ji v odvození podle $\mathscr{G}$ aplikací tohoto pravidla nahradit.

Příklad 3.3.4. Ke gramatice

$$S \to A \mid B,$$
$$A \to aaA \mid bba,$$
$$B \to bA \mid baB \mid e$$

se konstrukcí z důkazu 3.3.3 sestrojí ekvivalentní gramatika

$$S \to A \mid B,$$
$$A \to aA_2 \mid bC_1,$$
$$A_2 \to aA,$$
$$C_1 \to bC_2,$$
$$C_2 \to aC_3,$$
$$C_3 \to e,$$
$$B \to bA \mid bB_2 \mid e,$$
$$B_2 \to aB.$$

Věta 3.3.5. *Každý jazyk typu* 3 *je rozpoznatelný konečným automatem.*

Důkaz. Nechť $L = L(\mathscr{G})$ pro nějakou gramatiku $\mathscr{G}$ typu 3. Podle 3.3.3 můžeme předpokládat, že $\mathscr{G} = (\Pi, \Sigma, S, P)$ má pouze pravidla tvaru $X \to aY$, $X \to Y$, $X \to e$. Sestrojíme zobecněný nedeterministický konečný automat $\mathscr{A} = (Q, \Sigma, I, F)$ rozpoznávající $L(\mathscr{G})$. Tím bude důkaz uzavřen, protože podle 2.1.13 lze $\mathscr{A}$ převést na ekvivalentní konečný automat. Definujme

$$Q = \Pi,\ I = \{S\},\ F = \{Z; Z \to e \text{ je pravidlo z } P\},$$

$$\delta(Z, a) = \{Y; Z \to aY \text{ je v } P\}$$

pro libovolná $Z \in Q$, $a \in \Sigma$,

$$\delta(Z, e) = \{Y; Z \to Y \text{ je v } P\}$$

pro libovolná $Z \in Q$.

Potom zřejmě pro libovolné Z, $Y \in Q$, $u \in \Sigma^*$ platí, že

$$X \Rightarrow_{\mathscr{G}}^* uY,$$

právě když $(X, u) \vdash_{\mathscr{A}}^* Y$ (viz 2.1.11). Jelikož pro libovolné $w \in \Sigma^*$ platí, že

$$S \Rightarrow_{\mathscr{G}}^* w,$$

právě když $S \Rightarrow^* wY \Rightarrow w$, pro nějaké Y takové, že $Y \to e$ patří do P, je také

$$S \Rightarrow^* w \Leftrightarrow (S, w) \vdash_{\mathscr{A}}^* Y$$

pro nějaké $Y \in F$, tzn.

$$S \Rightarrow^* w,$$

právě když $w \in L(\mathscr{A})$.

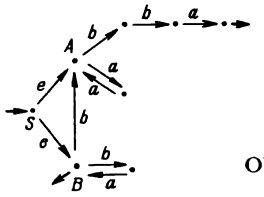
Ověřili jsme, že $L(\mathscr{A}) = L(\mathscr{G})$.

Příklad 3.3.6. Ke gramatice $\mathscr{G}$:

$$S \to A \mid B,$$
$$A \to aaA \mid bba,$$
$$B \to bA \mid baB \mid e$$

sestrojíme ekvivalentní zobecněný nedeterministický konečný automat.

Výchozí gramatika není ve tvaru, který předpokládá důkaz věty 3.3.5. Do tohoto tvaru je možno ji převést tak jako v př. 3.3.4. Z takto získané gramatiky je pak podle popsané konstrukce



Obr. 33

možno přejít k automatu znázorněnému na obr. 33. Čtenář se může sám přesvědčit, že k tomuto automatu lze přejít přímo od gramatiky $\mathscr{G}$.

Všimneme si ještě, že stejnou generativní sílu jako regulární gramatiky mají i bezkontextové gramatiky, u nichž se na pravé straně každého pravidla vyskytuje nejvýše jeden neterminál, a to pouze jako levý krajní symbol.

Definice 3.3.7. Gramatika $\mathscr{G}$ se nazývá *levá lineární gramatika*, jestliže má pouze pravidla typu $X \to Yw$ nebo $X \to w$, kde X, Y jsou neterminály a w řetěz terminálů.

Věta 3.3.8. *Jazyk je generovatelný levou lineární gramatikou, právě když je regulární.*

Důkaz. Budiž $\mathscr{G} = (\Pi, \Sigma, S, P)$ levá lineární gramatika. Definujme gramatiku $\mathscr{G}' = (\Pi, \Sigma, S, P')$ předpisem

$$X \to \alpha^{\mathrm{R}} \in P' \Leftrightarrow X \to \alpha \in P\,.$$

Potom $\mathscr{G}'$ je zřejmě pravá lineární gramatika neboli gramatika typu 3. Dále je vidět, že

$$L(\mathscr{G}') = [L(\mathscr{G})]^{\mathrm{R}}\,.$$

Tedy jazyk L je generovatelný levou lineární gramatikou, právě když L^{R} je typu 3, což je podle 2.2.17 právě tehdy, když L je regulární.

Z toho, co jsme zatím dokázali, dostáváme charakterizaci regulárních jazyků jakožto jazyků, které lze generovat gramatikami splňujícími dvě vlastnosti:

1. během libovolné derivace se v každém řetězu vyskytuje nejvýše jeden neterminál;
2. tento neterminál se vyskytuje vždy na pravém (resp. levém) konci slova.

Uveďme příklad gramatik porušujících některou z obou podmínek a generujících už jazyk, který není regulární.

Příklad 3.3.9. 1. Gramatika

$$S \to aSB \mid e\,,\ B \to b$$

generuje jazyk $L = \{a^n b^n;\ n \geqq 0\}$, který podle 1.3.6 není regulární.

2. Gramatika

$$S \to aS_1 \mid e\,,\ S_1 \to Sb$$

generuje tentýž jazyk L.

Druhá z gramatik kombinuje pravidla přípustná u levé lineární gramatiky s pravidly přípustnými u pravé lineární gramatiky a generuje už jazyk, který není regulární. Ukážeme, že takové gramatiky už mohou generovat všechny tzv. lineární jazyky.

Definice 3.3.10. *Gramatika* se nazývá *lineární*, jestliže obsahuje pouze pravidla tvaru $X \to uYv$ nebo $X \to u$, kde X, Y jsou neterminály a u, v řetězy terminálů.

Jazyk se nazývá *lineární*, jestliže je jej možno generovat některou lineární gramatikou.

Lemma 3.3.11. *Každý lineární jazyk je možno generovat gramatikou, která má pouze pravidla tvaru $X \to uY$ nebo $X \to Yu$ nebo $X \to u$, kde X, Y jsou neterminály a u řetěz terminálů.*

Důkaz. Libovolnou lineární gramatiku $\mathscr{G} = (\Pi, \Sigma, S, P)$ změníme na gramatiku v popsaném tvaru tak, že každé pravidlo tvaru $X \to uYv$ nahradíme dvojicí pravidel

$$X \to uZ\,,\ Z \to Yv\,,$$

kde Z je nově přidaný, v jiných pravidlech se nevyskytující neterminál. Ostatní pravidla zůstanou beze změny.

Důsledek 3.3.12. *Třída regulárních jazyků je vlastní podtřídou lineárních jazyků.*

Důkaz. Plyne okamžitě z 3.3.9 tvrzení 2 a 3.3.1 tvrzení 1.

ROZŠIŘUJÍCÍ ČÁST

3.4. Kategoriální gramatiky

Kategoriální gramatiky (viz čl. [5]) jsou dalším z teoretických nástrojů navržených ke zkoumání přirozených jazyků. Byly zavedeny přibližně ve stejné době jako generativní gramatiky, na rozdíl od nich se však mimo rámec lingvistiky dosud podstatněji neuplatnily.

Definice 3.4.1. Budiž A konečná abeceda (*množina primitivních kategorií*). *Množinou kategorií nad* A nazýváme nejmenší množinu slov $K(A)$ takovou, že

1. $A \subseteq K(A)$,
2. jestliže slova α a β patří do $K(A)$, potom slova (α/β) a $(\alpha\backslash\beta)$ patří do $K(A)$.

Příklad 3.4.2. $(a/(a/a))$, $((a\backslash b)/a)$, $(((b/a)/b)\backslash(a\backslash a))$ jsou kategorie nad $\{a, b\}$.

Definice 3.4.3. Řetězy kategorií zjednodušujeme těmito dvěma *redukčními pravidly*:

(1) $\quad (\alpha/\beta)\,\beta \to \alpha$,

(2) $\quad \beta(\beta\backslash\alpha) \to \alpha$,

kde α, β označují libovolné kategorie.

Příklad 3.4.4. Řetěz kategorií

$$(a/(b\backslash a))\,(b\backslash a)\,(a\backslash b)$$

lze podle pravidla (1) zredukovat na $a(a\backslash b)$ a dále podle (2) na b.

Definice 3.4.5. *Kategoriální gramatikou* budeme nazývat každou čtveřici $\mathscr{K} = (\Sigma, A, f, s)$, kde Σ je konečná abeceda, A je konečná množina (primitivních kategorií), $s \in A$ (cílová kategorie) a f je zobrazení, které každému prvku $x \in \Sigma$ přiřazuje konečnou množinu kategorií nad A, tzn. $f\colon \Sigma \to P_{\text{fin}}(K(A))$.

Příklad 3.4.6.

$$\mathscr{K} = (\Sigma, A, s, f),$$

kde

$$\Sigma = \{0, 1, 2\}, \quad A = \{a, b, s\}$$

a

$$f(0) = \{a, (s\backslash(a\backslash s))\},$$
$$f(1) = \{b, (s\backslash(b\backslash s))\},$$
$$f(2) = \{s\},$$

je kategoriální gramatika.

Každá kategoriální gramatika $\mathscr{K} = (\Sigma, A, s, f)$ umožňuje přiřadit libovolnému slovu ze Σ^+ množinu řetězů kategorií tímto předpisem:

$$f(a_1 a_2 \dots a_n) = f(a_1)\, f(a_2) \dots f(a_n).$$

Tak gramatika z př. 3.4.6 přiřazuje slovu 021 řetězy kategorií

$$asb,$$
$$as(s\backslash(b\backslash s)),$$
$$(s\backslash(a\backslash s))\, sb,$$
$$(s\backslash(a\backslash s))\, s(s\backslash(b\backslash s)).$$

Definice 3.4.7. *Jazyk $L(\mathscr{K})$ reprezentovaný kategoriální gramatikou* $\mathscr{K} = (\Sigma, A, s, f)$ je tvořen všemi slovy $w \in \Sigma^+$, kterým zobrazení f přiřazuje nějaký řetěz kategorií

$$\alpha_1 \alpha_2 \dots \alpha_n \in f(w),$$

který lze redukčními pravidly (podle 3.4.3) převést na cílovou kategorii s.

Příklad 3.4.8. Gramatika $\mathscr{K}$ z př. 3.4.6 reprezentuje jazyk

$$L = \{w2w^R; w \in \{0, 1\}^*\},$$

jak může čtenář sám ověřit. Jenom poznamenejme, že např. slovu 01210 patřícímu do L lze přiřadit kategorii

$$abs(s\backslash(b\backslash s))\,(s\backslash(a\backslash s)),$$

kterou lze postupně redukovat takto:

$$\begin{aligned} &abs(s\backslash(b\backslash s))\,(s\backslash(a\backslash s)) \Rightarrow \\ &\Rightarrow ab(b\backslash s)\,(s\backslash(a\backslash s)) \Rightarrow \\ &\Rightarrow as(s\backslash(a\backslash s)) \Rightarrow a(a\backslash s) \Rightarrow s\,. \end{aligned}$$

Naproti tomu žádná z kategorií přiřazených např. slovu 021 se neredukuje na s, což je v souhlasu s tím, že $021 \notin L$.

3.4.9. (Návrh kategoriální gramatiky.)

Schopnost popsat nějaký předem daný jazyk pomocí určitého formalismu se získává cvikem. Po prostudování příkladů gramatik a teoretických nástrojů potřebných pro jejich transformace uvedených v této knize získá čtenář alespoň základní zběhlost v užívání generativních gramatik. Návrh kategoriálních gramatik však vyžaduje poněkud odlišný způsob myšlení, a proto mu věnujme alespoň několik následujících řádků. Budeme přitom užívat slova „kategorie" zprvu v intuitivním smyslu, a to ve fázi, kdy budeme zjišťovat, že v daném jazyce jsou různé kategorie symbolů a tyto kategorie budeme verbálně vymezovat. (Symbol může být několika různých kategorií.) Jednotlivé kategorie vymezíme jejich vzájemnými vztahy. Teprve zachycení těchto vztahů vede k formálnímu zápisu kategorií, který známe z definice 3.4.1.

Při vytváření kategoriální gramatiky popisující určitý daný jazyk se snažíme najít všechny role, které mohou jednotlivé symboly abecedy hrát ve správně utvořených slovech, určit jejich vzájemné vztahy a v souhlase s tím určit kategorie symbolů.

Tak např. v jazyce

$$L = \{w2w^R; w \in \{0, 1\}^*\}$$

jsou nutně dvě kategorie výskytů u každého ze symbolů 0, 1. Do

jedné kategorie spadají výskyty nalevo od symbolu 2, do druhé výskyty napravo od 2. Samotný symbol 2 patří do L, a proto jedna z kategorií, které budou přiřazeny 2, musí být cílová kategorie s.

Označme prozatímně vytypované kategorie a, a', b, b', s.

0 je kategorie a, leží-li nalevo od 2,
0 je kategorie a', leží-li napravo od 2,
1 je kategorie b, leží-li nalevo od 2,
1 je kategorie b', leží-li napravo od 2,
2 je kategorie s.

Zjistěme vztahy mezi získanými kategoriemi. Jestliže ke správně utvořenému slovu (tj. slovu kategorie s) připojíme zleva i zprava symbol 0, vznikne opět slovo z jazyka L. Proto by měla být možná redukce

$$(*) \qquad asa' \Rightarrow^* s\,.$$

Tento vztah umožňuje vyjádřit kategorii a' pomocí a a s. Redukce $(*)$ bude možná, bude-li např. řetěz sa' kategorie $(a\backslash s)$, jak se můžeme přesvědčit dosazením. Řetěz sa' pak bude kategorie $(a\backslash s)$, bude-li a' kategorie $(s\backslash(a\backslash s))$. Tím jsme pro symbol 0 získali kategorie a a $(s\backslash(a\backslash s))$. Podobně z definice jazyka L plyne, že má být možná redukce $bsb' \Rightarrow^* s$. Odtud podobně jako u $(*)$ nahradíme b' kategorií $(s\backslash(b\backslash s))$. Takto jsme získali gramatiku z př. 3.4.6.

Získaná gramatika není jedinou možnou reprezentací jazyka L. Symbolu 0 bychom mohli přiřadit dvojici kategorií (a/s), $(a\backslash s)$, apod.

Třída kategoriálních gramatik je zajímavá zejména pro odlišný přístup k popisu jazyka. Jejich generativní sílu je možné vyjádřit v termínech Chomského hierarchie.

Věta 3.4.10. [5]. *Jazyk je reprezentovatelný kategoriální gramatikou, právě když je bezkontextový a neobsahuje prázdné slovo.*

Důkaz věty, který je značně rozsáhlý, může čtenář najít v čl. [5].

Zavedení tohoto typu gramatik bylo inspirováno mechanismem generování maker při překladu programovacích jazyků.

Tvar makrogramatik, se kterým se zde seznámíme, je oproti původní definici (v práci [10]) poněkud zjednodušen. I v této verzi jsou však makrogramatiky podstatným zesílením aparátu bezkontextových gramatik. Povahu tohoto „zesílení" si asi nejlépe uvědomíme, když budeme na neterminály v bezkontextových gramatikách pohlížet jako na procedury generující řetězy. Pravidla gramatik pak popisují způsob generování, při kterém ovšem procedura může vyvolat procedury další. Tak např. pravidlo $A \to aBb$ znamená, že procedura A vygeneruje symbol a, za kterým bude následovat nějaký řetěz vygenerovaný procedurou B a slovo bude končit symbolem b. U makrogramatik mohou tyto procedury navíc záviset na parametrech.

Příklad 3.5.1. Ukažme, že makrogramatika zadaná pravidly

(1) $S \to A(e, e)$,

(2) $A(x, y) \to A(ax, yxxa)$,

(3) $A(x, y) \to y$

generuje jazyk $L = \{a^{n^2}; n \geqq 0\}$.

Prohlédněme si jednotlivá pravidla. V závorkách za neterminály jsou uvedeny parametry. Neterminál S je bez parametrů, A má dva parametry. Na místě parametrů mohou být řetězy tvořené terminálními symboly a proměnnými. V našem případě je a terminál, x, y jsou proměnné. Hodnoty parametrů na pravé straně pravidla ( ) se vypočtou z hodnot parametrů na levé straně. Pro ilustraci odvoďme slovo a^4:

$$S \Rightarrow^{(1)} A(e, e) \Rightarrow^{(2)} A(a, a) \Rightarrow^{(2)} A(aa, aaaa) \Rightarrow^{(3)} aaaa\,.$$

Teď už je také vidět princip činnosti této gramatiky. Pomocí pravidla (2) [sestaveného na základě vztahu $(k + 1)^2 = k^2 + 2k + 1$] se pro jednotlivá n postupně vytvářejí dvojice (a^n, a^{n^2}). Pravidlem (3) je možno oddělit druhou složku, a tím odvození končí.

Příklad 3.5.2. 1. Jazyk

$$L_1 = \{\S(w\S)^n;\ w \in \{0,1\}^*, n \geqq 1\}$$

je generován gramatikou

$$S \to A(e),$$
$$A(x) \to A(x0) \mid A(x1) \mid \S B(x),$$
$$B(x) \to B(x)\,B(x) \mid x\S.$$

2. Jazyk

$$L_2 = \{\S w_1 \S w_2 \S \ldots \S w_n \S w_i;\ n \geqq 1,\ 1 \leqq i \leqq n,$$
$$\text{pro všechna } j,\ w_j \in \{0,1\}^*\},$$

je generován gramatikou

$$S \to F(e),$$
$$F(x) \to F(x0) \mid F(x1) \mid G(e)\,\S x G(e)\,\S x,$$
$$G(x) \to G(x)\,G(x) \mid G(x0) \mid G(x1) \mid \S x \mid e.$$

Zajímavým speciálním případem makrogramatik jsou tzv. *registrové gramatiky*, které se vyznačují těmito dvěma rysy:

1. Na pravé straně každého pravidla je nejvýše jeden neterminál.

2. Všechny neterminály mají stejný počet parametrů s výjimkou počátečního symbolu, který je bez parametrů.

Gramatika z př. 3.5.1 je registrovou gramatikou.

Každé pravidlo registrové gramatiky lze realizovat jako paralelní přiřazení nového obsahu jistému konečnému počtu registrů. Registry obsahují řetězy a nový obsah registru lze získat zřetězením dosavadních obsahů registrů a jistých pevně daných řetězů. Tak např. příkaz (2) z př. 3.5.1 je realizován paralelním přiřazením

$$x := ax,$$
$$y := yxxa.$$

3.6. Gramatiky s řízeným přepisováním

Význačným rysem gramatik, jak jsme je zavedli v čl. 3.1, je naprostý nedeterminismus při odvozování. Někdy je ale přirozenější používat modelů gramatik, u nichž je výběr pravidel během generování nějakým způsobem řízen.

Tak např. u *maticových gramatik* (čl. [23]) jsou zadány posloupnosti pravidel, která se během odvozování musí vyskytovat pohromadě. Každá aplikace takové posloupnosti pravidel (matice) začíná prvním pravidlem posloupnosti a pokračuje postupně všemi dalšími pravidly až k pravidlu poslednímu. Teprve potom je možné aplikovat některou další matici.

Příklad 3.6.1. Maticová gramatika zadaná maticemi

$$\langle S \to AC \rangle,$$
$$\langle A \to aAb;\ C \to cC \rangle,$$
$$\langle A \to e;\ C \to e \rangle,$$

v níž S je počáteční symbol a a, b, c jsou terminály, generuje jazyk

$$L = \{a^n b^n c^n;\ n \geqq 0\}.$$

O tomto jazyku dokážeme v kap. 4, že není bezkontextový. Jelikož všechna jednotlivá pravidla použité gramatiky jsou bezkontextová, je vidět, že seskupování pravidel do matic může zvýšit generativní sílu určitého typu gramatik.

O jiný způsob vzájemné vazby mezi pravidly se opírají *uspořádané gramatiky*. U nich je množina pravidel částečně uspořádána a pravidlo je možné aplikovat pouze v případě, že nelze aplikovat žádné pravidlo, které je v uspořádání předchází.

Příklad 3.6.2. Gramatika zadaná pravidly

(1) $S \to AC$,
(2) $A \to aBb$,
(3) $C \to cD$,
(4) $B \to A$,
(5) $D \to C$,

(6) $C \to e$,

(7) $D \to e$,

(8) $A \to e$

s částečným uspořádáním pravidel

$$(2) < (3),\ (4) < (5),\ (6) < (7) < (8)$$

generuje jazyk

$$\{a^n b^n c^m;\ 0 \leqq m \leqq n\}.$$

Tento jazyk není bezkontextový.

Během odvozování v této gramatice může být pravidlo (3) použito až poté, kdy se pravidlem (2) A přepsalo na aBb. Další použití pravidla (3) je možné až po aplikaci pravidla (5). To je ovšem možné až po tom, kdy pravidlo (4) připravilo k použití pravidlo (2). Promyslete význam uspořádání posledních tří pravidel.

Podrobnější informace o obou zmíněných typech gramatik i dalších gramatikách s řízeným přepisováním, jakými jsou např. programované gramatiky, lze najít v knize [28].

3.7. L-systémy

Všechny typy gramatik, o kterých jsme se zmínili, mají přes všechny odlišnosti jeden velmi podstatný rys společný. U všech se předpokládá, že v každém kroku se přepisuje jen jeden omezený úsek řetězu. Takový mechanismus dobře vystihuje odvozování v různých formálních systémech.

Zcela jiný přístup vyžadují generativní systémy motivované biologicky. Představme si idealizovaný jednoduchý organismus tvořený lineárním řetězem buněk, ve kterém se vždy po určitém okamžiku (pro všechny buňky stejném) každá buňka rozdělí na dvě. Vývoj tohoto organismu z jediné buňky c lze znázornit posloupností $c, cc, cccc, cccccccc, \ldots$. Na tuto posloupnost můžeme pohlížet jako na odvozování řetězů ze slova c pomocí pravidla $c \to cc$. Toto pravidlo je ovšem v každém okamžiku aplikováno na všechny symboly současně.

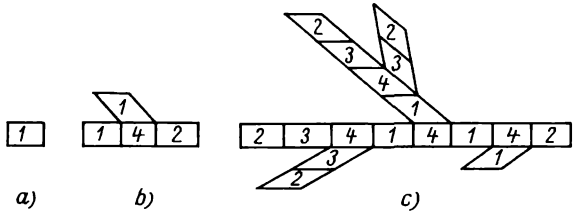
Příklad 3.7.1. Přepisovacími pravidly můžeme za pomoci závorek reprezentovat i růst větvících se struktur. Například soustava pravidel

$$1 \to 23,$$
$$2 \to 1,$$
$$3 \to 4(1)2,$$
$$4 \to 4,$$
$$(\to (,$$
$$) \to)$$

dává (při uvedeném paralelním přepisování) postupně řetězy

$$1 \Rightarrow 23 \Rightarrow 14(1)2 \Rightarrow 234(23)1 \Rightarrow$$
$$\Rightarrow 14(1)24(14(1)2)23 \Rightarrow$$
$$\Rightarrow 234(23)14(234(23)1)14(1)2 \Rightarrow \ldots.$$

Budeme-li opět každou z číslic chápat jako buňku v určitém stavu a výraz v závorce za symbolem jako větev vyrůstající z příslušné buňky, je možné postup odvozování zachytit na obr. 34. (Střídavé větvení, které na tomto obrázku používáme, ovšem z tvaru slova neplyne.) (a) označuje začátek, (b) třetí člen odvození a (c) šestý člen odvození.



Obr. 34

Bylo to právě na popisu zákonitostí růstu jednoho druhu chaluh, kde Lindenmayer předvedl použití podobného systému přepisovacích pravidel. Dnes je tento typ systémů na jeho počest nazýván L-systémy. Uvedeme definici jejich speciálního, ale velmi důležitého typu.

Definice 3.7.2. 0L-*systémem* budeme nazývat každou trojici $\mathcal{G} = (V, P, \sigma)$, kde V je konečná abeceda, P množina přepisovacích pravidel tvaru $a \to \alpha$ $(a \in V,\ \alpha \in V^*)$ a $\sigma \in V^+$. Bývá zvykem předpokládat úplnost množiny P, tzn. že pro každé $a \in V$ existuje alespoň jedno slovo $\alpha \in V^*$ takové, že $a \to \alpha \in P$.

V př. 3.7.1 je $V = \{1, 2, 3, 4,), (\}$ a $\sigma = 1$.

Nula v označení 0L signalizuje, že se jedná o L-systémy bez interakce, tzn. na přeměnu symbolu (buňky) nemají vliv sousední symboly. Jinými slovy, všechna pravidla mají tvar analogický jako v bezkontextových gramatikách. Obecnější typ L-systémů, systémy s interakcí, se označují jako IL-systémy. Těmi se zabývat nebudeme.

Definice 3.7.3. Slovo $u = a_1 \dots a_n \in V^+$ lze v 0L-systému

$$\mathcal{G} = (V, P, \sigma)$$

bezprostředně přepsat na slovo v, jestliže existují pravidla

$$a_1 \to \alpha_1, \dots, a_n \to \alpha_n \in P$$

taková, že $\alpha_1 \dots \alpha_n = v$. Značíme to $u \to v$. Píšeme $w \Rightarrow^* z$, jestliže slovo w lze v konečně mnoha krocích přepsat na z.

Jazykem generovaným systémem $\mathcal{G}$ nazýváme jazyk

$$L(\mathcal{G}) = \{w;\ w \in V^* \text{ a } \sigma \Rightarrow^* w\}.$$

O L-systémech dnes existuje rozsáhlá, zejména časopisecká literatura. Autoři se v ní nezřídka věnují speciálním tvarům a modifikacím 0L-systémů a jejich kombinacím. Pro rychlejší orientaci většinou používají systém zkratek umožňujících identifikovat, o jaký typ systému se jedná. Proto se můžeme setkat s články, které již ve svých nadpisech ohlašují, že se zabývají např. ET0L-systémy či PD0L-systémy.

Vysvětlíme, jak tyto názvy dešifrovat. Skupina posledních dvou symbolů – 0L, resp. IL – určuje, zda se jedná o systémy bez interakce, resp. s interakcí. Před nimi se mohou vyskytovat některé ze symbolů D, P, E, T. Jejich význam je tento:

D: označuje deterministický systém, tj. mezi přepisovacími

pravidly neexistují dvě různá pravidla se stejnou levou stranou. V př. 3.7.1 jsme se tedy setkali s D0L-systémem.

P: (z anglického propagating) označuje rozpínavé systémy. U těchto systémů není na pravé straně žádného pravidla prázdné slovo. Příklad 3.7.1 je proto také P0L-systémem. Jelikož je navíc deterministický, je možné jej označit jako PD0L-systém.

E: (z anglického extended) označuje rozšířené systémy. Pro L-systémy je charakteristické, že abeceda V není dělena na neterminály a terminály. Při každém kroku přepisování se s každým symbolem něco děje (např. to, že zůstává stejný). V některých aplikacích je ale užitečné zaměřit se pouze na ta odvoditelná slova, jejichž všechny symboly mají určitou význačnou vlastnost. E0L-systém je proto definován jako čtveřice

$$\mathcal{G} = (V, P, \sigma, \Sigma),$$

kde (V, P, σ) je 0L-systém a $\Sigma \subseteq V$. Jazyk reprezentovaný tímto systémem je pak

$$L(\mathcal{G}) = \{w;\ w \in \Sigma^* \text{ a } \sigma \Rightarrow^* w\}.$$

T: označuje tabulkový 0L-systém. Místo jediného souboru P přepisovacích pravidel může mít tento typ systémů několik takových souborů $P_1, \ldots, P_k$. Při každém kroku přepisování se pak smějí aplikovat pouze pravidla patřící do téhož souboru.

Občas je možné se setkat se zkratkami obsahujícími i další písmena, např. C, G, S, U. Jejich užití je ovšem méně obvyklé, zpravidla se užívá pouze lokálně v článcích a neproniká až do jejich nadpisů.

V souvislosti s tabulkovými systémy je užitečné si uvědomit, že jejich užití se nemusí omezit pouze na generování jazyků, ale lze jimi reprezentovat i překlad z jednoho jazyka do druhého. Jestliže totiž označíme jednotlivé tabulky systému pomocí symbolů z nějaké konečné abecedy Δ, potom každý řetěz $u \in \Delta^*$ můžeme chápat jako předpis určující, jaké tabulky a v jakém pořadí se mají při generování slova používat. Předepsaným způsobem vznikají slova jistého jazyka $L_u \subseteq V^*$, kde V je abeceda systému. Daný systém tedy určuje jisté zobrazení $\Delta^* \to P(V^*)$.

V případě, že systém je navíc deterministický, určuje zobrazení $\Delta^* \to V^*$.

Příklad 3.7.4. Mějme EDT0L-systém (výstižněji: PEDT0L-systém)

$$\mathcal{G} = (V, P, \sigma, \Sigma),$$

kde

$$V = \{a, b, p, q\}, \quad \Sigma = \{a, b\}, \quad \sigma = pq$$

a P je množina těchto tří tabulek přepisovacích pravidel:

$$0\begin{cases} p \to apa, \\ q \to aq, \\ a \to a, \\ b \to b, \end{cases}$$

$$1\begin{cases} p \to bpb, \\ q \to qb, \\ a \to a, \\ b \to b, \end{cases}$$

$$2\begin{cases} p \to bb, \\ q \to b, \\ a \to b, \\ b \to a. \end{cases}$$

Tento systém převádí např. slovo 0112 na slovo *baabbaabbbaa* patřící do jazyka $L(\mathcal{G})$ takto: po aplikaci tabulky 0 vzniká z pq slovo $apaaq$, po následující aplikaci tabulky 1 se přepíše na slovo $abpbaaqb$ atd.

ET0L-systémům se v mnohém podobají stromové automaty, se kterými se setkáme v kap. 7.

Čtenáři, který by se s L-systémy rád seznámil podrobněji, lze doporučit knihu [28].

Zajímavé je také srovnání složitosti L-systémů v čl. [25], [11]. V prvním z nich je popsán jednoduchý PD0L-systém modelující růst chaluhy, zatímco ve druhém je to mnohem komplikovanější systém modelující růst složnokvěté rostliny.

4. ZÁKLADY TEORIE BEZKONTEXTOVÝCH JAZYKŮ

ZÁKLADNÍ ČÁST

Značná pozornost, kterou budeme v této a následující kapitole věnovat bezkontextovým jazykům, odpovídá jejich praktické důležitosti. Pomocí bezkontextových jazyků je možné z větší části definovat syntaxi vyšších programovacích jazyků. V posledních dvaceti letech (počínaje jazykem Algol 60) se takové zadávání syntaxe jazyka stalo téměř pravidlem. Proto také vztah bezkontextových gramatik a zásobníkových automatů, o nichž budeme pojednávat, slouží v současné době jako východisko různých metod konstrukce základních částí kompilátorů.

V této kapitole půjde zejména o pochopení základních myšlenek, o které se opírá přechod od bezkontextové gramatiky k ekvivalentnímu zásobníkovému automatu. Detailnějšímu rozboru praktických metod tohoto přechodu se budeme věnovat v následující kapitole.

V této kapitole probereme i několik dalších výsledků bohatě rozpracované matematické teorie bezkontextových jazyků, které poskytují vhodný aparát k úpravám bezkontextových gramatik a ke zkoumání, zda daný jazyk je bezkontextový, či nikoliv.

4.1. Redukované gramatiky

K užitečnému zjednodušení úvah, ale i praktických konstrukcí, vede možnost převést každou bezkontextovou gramatiku na ekvivalentní redukovanou gramatiku.

Definice 4.1.1. Bezkontextová gramatika $\mathscr{G} = (N, \Sigma, S, P)$ taková, že $L(\mathscr{G}) \neq \emptyset$, se nazývá *redukovaná*, právě když jsou splněny tyto dvě podmínky:

(1) Ke každému neterminálu X existuje alespoň jedno terminální slovo w takové, že $X \Rightarrow_{\mathscr{G}}^{*} w$.

(2) Ke každému neterminálu $X \neq S$ existují slova

$$\alpha, \beta \in (\Pi \cup \Sigma)^*$$

taková, že $S \Rightarrow_{\mathscr{G}}^* \alpha X \beta$.

Redukovaná gramatika tedy neobsahuje zbytečné neterminály, tj. neterminály, které se nemohou vyskytnout během žádného odvození terminálního řetězu ze symbolu S. Ke gramatice $\mathscr{G}$ takové, že $L(\mathscr{G}) = \emptyset$, pochopitelně není možné sestrojit ekvivalentní redukovanou gramatiku, protože z každého neterminálu redukované gramatiky je možné odvodit nějaký terminální řetěz. To platí i pro počáteční neterminál, a proto je jazyk generovaný libovolnou redukovanou gramatikou nutně neprázdný.

Lemma 4.1.2. *Ke každé bezkontextové gramatice $\mathscr{G}$ takové, že $L(\mathscr{G}) \neq \emptyset$, lze sestrojit ekvivalentní redukovanou gramatiku.*

Důkaz. Popišme algoritmus redukce. Budiž

$$\mathscr{G} = (\Pi, \Sigma, S, P)$$

výchozí bezkontextová gramatika.

1. V první fázi konstrukce sestrojme množinu

$$V = \Sigma \cup \{X \in \Pi;\ \text{existuje } w \in \Sigma^* \text{ takové, že } X \Rightarrow_{\mathscr{G}}^* w\}$$

a zároveň ověřme, zda $L(\mathscr{G}) = \emptyset$. Množinu V vytváříme tak, že do ní nejprve vložíme všechny terminály a postupně do ní přidáváme všechny neterminály, které lze v jediném kroku přepsat na řetěz složený ze symbolů, o nichž už máme zjištěno, že patří do V. Formálně řečeno: definujme

$$V_0 = \Sigma,$$

$$V_{i+1} = V_i \cup \{X \in \Pi;\ \text{existuje } \alpha \in V_i^* \text{ takové, že } X \to \alpha \in P\}.$$

Zřejmě

$$V_0 \subseteq V_1 \subseteq V_2 \subseteq \ldots \subseteq \Sigma \cup \Pi.$$

Existuje proto n takové, že $V_n = V_{n+1}$, a tedy, jak plyne z defi-

nice V_{i+1}, je $V_n = V_{n+k}$ pro libovolné $k \geq 1$. Je vidět, že $V = V_n$, a můžeme se přesvědčit, zda $L(\mathscr{G}) \neq \emptyset$, neboť

$$L(\mathscr{G}) \neq \emptyset \Leftrightarrow S \in V.$$

Jakmile je množina V sestrojena, odstraníme z gramatiky $\mathscr{G}$ všechna pravidla, v nichž se na levé nebo pravé straně vyskytuje neterminál nepatřící do V. Takto dostaneme jistou gramatiku $\mathscr{G}'$. Určitě je $L(\mathscr{G}') \subseteq L(\mathscr{G})$, neboť každá derivace podle $\mathscr{G}'$ je také derivací podle $\mathscr{G}$.

Ukážeme, že také $L(\mathscr{G}) \subseteq L(\mathscr{G}')$. Předpokládejme opak, tj. že existuje slovo $w \in L(\mathscr{G}) - L(\mathscr{G}')$. Slovo w lze odvodit podle $\mathscr{G}$, ale ne podle $\mathscr{G}'$. Musí proto každé odvození w podle $\mathscr{G}$ obsahovat symbol z $\Pi - V$, tzn. $S \Rightarrow_{\mathscr{G}}^* \alpha X \beta \Rightarrow^* w$ pro nějaká α, β a $X \in \Pi - V$. Potom ale $X \Rightarrow_{\mathscr{G}}^* w_1$ pro nějaké podslovo terminálního řetězu w, což je ve sporu s předpokladem, že $X \in \Pi - V$. Předpoklad $L(\mathscr{G}) - L(\mathscr{G}') \neq \emptyset$ jsme dovedli ke sporu. Je proto skutečně $L(\mathscr{G}) \subseteq L(\mathscr{G}')$.

Dohromady jsme dostali $L(\mathscr{G}) = L(\mathscr{G}')$.

Přesvědčíme se, že gramatika $\mathscr{G}'$ splňuje podmínku (1) z definice 4.1.4 redukované gramatiky. Nechť X je libovolný neterminál z $\mathscr{G}'$. Potom $X \in V$ podle konstrukce gramatiky $\mathscr{G}'$. To znamená, že $X \Rightarrow_{\mathscr{G}}^* w$ pro nějaké $w \in \Sigma^*$. Všechny neterminály vyskytující se v tomto odvození se nakonec přepíší na terminální řetěz, tzn. všechny patří do V. Z toho ovšem plyne, že žádné přepisovací pravidlo užité při odvození $X \Rightarrow^* w$ se při konstrukci $\mathscr{G}'$ z množiny pravidel nevyřadí. Je tedy toto odvození také odvozením podle $\mathscr{G}'$. Tím jsme ověřili, že každý neterminál gramatiky $\mathscr{G}'$ lze v $\mathscr{G}'$ přepsat na řetěz terminálů.

2. Ve druhé fázi konstrukce přejdeme od gramatiky $\mathscr{G}'$ k ekvivalentní gramatice $\mathscr{G}''$, která už bude redukovaná. Při tomto přechodu odstraníme všechny neterminální symboly, které se nemohou vyskytnout v žádném řetězu odvoditelném z S. Nejprve sestrojíme množinu

$$U = \{X \in \Pi;\ S \Rightarrow_{\mathscr{G}}^* \alpha X \beta \text{ pro nějaká } \alpha, \beta \in (\Pi \cup \Sigma)^*\}$$

induktivně tak, že do U bude patřit S a postupně jsou přidávány všechny neterminály, které se vyskytují na pravé straně nějakého

$U_0 = \{S\}$

$U_{i+1} = U_i \cup \{X \in \Pi$

pravidla $Y \to \gamma$, kde Y je neterminál, o němž jsme už zjistili, že patří do U. Konstrukce U je skončena, jakmile už není možné připojit žádný nový neterminál.

Gramatiku $\mathscr{G}''$ nyní z $\mathscr{G}'$ dostaneme vypuštěním všech pravidel, na jejichž levé nebo pravé straně se vyskytuje nějaký neterminál nepatřící do U. Obdobně jako v první části důkazu se ukáže, že $L(\mathscr{G}'') = L(\mathscr{G}')$ a že $\mathscr{G}''$ splňuje podmínku (2) z definice 4.1.4. Je ovšem třeba se ještě ujistit, že při přechodu od gramatiky $\mathscr{G}'$ ke $\mathscr{G}''$ nepřestala platit podmínka (1) z téže definice. Pro každý neterminál X gramatiky $\mathscr{G}''$ platí podle první části důkazu, že

(*) $\quad X \Rightarrow_{\mathscr{G}'}^{*} w$ pro nějaký terminální řetěz w.

Navíc X patří do množiny U, a proto

$$S \Rightarrow_{\mathscr{G}''}^{*} \alpha X \beta \Rightarrow_{\mathscr{G}'}^{*} \alpha w \beta ,$$

pro vhodná α, β. Každý neterminál vyskytující se v libovolném odvození odpovídajícím (*) lze tedy odvodit z S a z toho plyne, že patří do U. To znamená, že odvození odpovídající (*) je také odvozením v $\mathscr{G}''$, a proto gramatika $\mathscr{G}''$ splňuje i podmínku (1) a je redukovaná.

Prohození obou fází algoritmu redukce nemusí obecně vést k redukované gramatice, jak ukážeme na příkladu.

Příklad 4.1.3. Gramatika zadaná pravidly

$$S \to aA \mid ab ,$$

$$A \to BC ,$$

$$B \to ba ,$$

$$D \to AB \mid e$$

přejde po skončení první fáze algoritmu redukce na gramatiku

$$S \to ab ,$$

$$B \to ba ,$$

$$D \to e ,$$

splňující podmínku (1) definice redukované gramatiky a po provedení druhé fáze dá redukovanou gramatiku $S \to ab$.

Při prohození obou fází nejprve dostaneme gramatiku

$$S \to aA \mid ab,$$

$$A \to BC,$$

$$B \to ba,$$

která splňuje podmínku (2), a poté dostaneme gramatiku

$$S \to ab,$$

$$B \to ba,$$

která pochopitelně splňuje podmínku (1), ale platnost podmínky (2) byla porušena odstraněním pravidel $S \to aA$ a $A \to BC$. Výsledná gramatika není redukovaná.

Z první části důkazu 4.1.2 okamžitě plyne následující tvrzení.

Věta 4.1.4. *Existuje algoritmus, který pro každou bezkontextovou gramatiku $\mathscr{G}$ rozhoduje, zda $L(\mathscr{G}) = \emptyset$.*

Redukce konečných automatů popsaná v čl. 1.5 vedla k nejmenšímu možnému konečnému automatu reprezentujícímu určitý jazyk. Lemma 4.1.2 k žádnému takovému výsledku nevede. V kap. 6 se seznámíme s důvody, proč tomu tak ani nemůže být. I když tedy algoritmus 4.1.2 nemusí vést k vůbec nejmenší gramatice reprezentující daný jazyk, přesto jeho užití vede často k radikálnímu zmenšení gramatiky, zvláště u gramatik vznikajících některými mechanickými transformacemi, které v dalším výkladu popíšeme.

4.2. Kanonické derivace, derivační stromy, jednoznačné gramatiky

Podstatné zjednodušení úvah o odvozování slov podle bezkontextové gramatiky je dáno možností omezit se pouze na tzv. *kanonické derivace*, tj. levé nebo pravé derivace.

Definice 4.2.1. Nechť $\mathcal{G}$ je bezkontextová gramatika. Řekneme, že řetěz α lze přepsat na řetěz β podle $\mathcal{G}$ *levým přepsáním*, jestliže existuje pravidlo $X \to \gamma$ z $\mathcal{G}$ takové, že $\alpha = \xi X \eta$, $\beta = \xi\gamma\eta$, přičemž ξ je nějaký řetěz terminálů a η je libovolný řetěz. Jestliže obráceně η je řetězem terminálů a ξ je libovolný řetěz, říkáme, že β vzniká z α *pravým přepsáním*. Odvození

$$\alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_r$$

se nazývá *levá derivace*, právě když pro každé $i = 0, \ldots, r-1$ vzniká α_{i+1} z α_i levým přepsáním. Odvození, ve kterém se vyskytují pouze pravá přepsání, se nazývá *pravá derivace*.

Levá derivace je tedy takové odvození, ve kterém se vždy přepisuje nejlevější neterminální symbol. V *pravém odvození* se naopak přepisuje vždy nejpravější neterminál.

Lemma, které nyní vyslovíme, je jednoduché, ale je vhodné si jeho důkaz důkladně promyslet. Myšlenkový obrat, o nějž se opírá, budeme dále velmi často používat i v komplikovanějších souvislostech.

Lemma 4.2.2. *Nechť $\mathcal{G} = (\Pi, \Sigma, S, P)$ je bezkontextová gramatika a nechť $X \Rightarrow_{\mathcal{G}}^{*} w$ pro nějaké $X \in \Pi$ a $w \in \Sigma^{*}$. Potom w lze z $\mathcal{G}$ odvodit nějakou levou derivací i nějakou pravou derivací.*

Důkaz. Dokážeme pouze první část lemmatu. Tvrzení o pravé derivaci se dokáže analogicky.

Podstata důkazu spočívá v tomto pozorování: Jestliže přepisujeme nějaký řetěz tvaru $\alpha X \beta$ na nějaký terminální řetěz w, potom u bezkontextové gramatiky nezávisí přepsání symbolu X na řetězech α, β ani na jejich dalším přepisování. Jinak řečeno, jestliže $\alpha X \beta \Rightarrow^{*} w$, potom w lze psát ve tvaru $v_1 u v_2$, kde $\alpha \Rightarrow^{*} v_1$, $X \Rightarrow^{*} u$, $\beta \Rightarrow^{*} v_2$. Jelikož se tedy části α, X i β přepisují nezávisle na sobě, je v derivaci $\alpha X \beta \Rightarrow^{*} w$ možné „přednostně“ provést aplikaci pravidel přispívajících k odvození $X \Rightarrow^{*} w$ a aplikace pravidel přispívající k $\alpha \Rightarrow^{*} v_1$ a $\beta \Rightarrow^{*} v_2$ až dodatečně. Z těchto důvodů lze aplikaci pravidel v odvození přeskupit tak, že nejprve se vždy přepisuje nejlevější neterminál. Tím vznikne levá derivace.

Podrobný důkaz vypadá takto:
Nechť

$$(*) \qquad X \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_r = w$$

je libovolné odvození podle gramatiky $\mathscr{G}$. Na základě tohoto odvození sestrojíme levou derivaci

$$(**) \qquad X \Rightarrow \beta_1 \Rightarrow \ldots \Rightarrow \beta_r = w \text{ podle } \mathscr{G}\,.$$

Postupujeme indukcí podle délky derivace $(*)$.

1. Nechť $r = 1$, potom $\alpha_1 = w$ a $X \to w \in P$. Položíme $\beta_1 = \alpha_1 = w$ a jsme hotovi.

2. Nechť umíme sestrojit levou derivaci $(**)$ na základě odvození $(*)$, jestliže $r \leqq k$, pro nějaké $k \geqq 1$. Ukážeme konstrukci pro $r = k + 1$:

$$\alpha_1 \notin \Sigma^*,$$

a tedy je možné psát je ve tvaru

$$\alpha_1 = \gamma_1 Y_1 \gamma_2 \ldots Y_s \gamma_{s+1},$$

kde

$$Y_i \in \Pi \text{ a } \gamma_i \in \Sigma^*.$$

Potom w lze psát ve tvaru

$$w = \gamma_1 w_1 \gamma_2 \ldots w_s \gamma_{s+1}$$

tak, že

$$(***) \qquad Y_i \Rightarrow^* w_i \quad \text{pro všechna } i,\ 1 \leqq i \leqq s\,.$$

Každé odvození $(***)$ má délku nejvýše k, a proto je lze podle indukčního předpokladu nahradit levým odvozením. Levé odvození $(**)$ nyní zkonstruujme takto: Položíme $\beta_1 = \alpha_1$, tj.

$$X \Rightarrow \gamma_1 Y_1 \gamma_2 \ldots Y_s \gamma_{s+1}$$

a v tomto odvození postupně provádíme levé derivace

$$Y_1 \Rightarrow^* w_1, \quad Y_2 \Rightarrow^* w_2, \ldots, Y_s \Rightarrow^* w_s,$$

tj.

$$X \Rightarrow \gamma_1 Y_1 \gamma_2 \dots Y_s \gamma_{s+1} \Rightarrow^*$$
$$\Rightarrow^* \gamma_1 w_1 \gamma_2 Y_2 \dots Y_s \gamma_{s+1} \Rightarrow^*$$
$$\Rightarrow^* \gamma_1 w_1 \gamma_2 w_2 \gamma_3 Y_3 \dots Y_s \gamma_{s+1} \Rightarrow^* \dots \Rightarrow^* w .$$

Příklad 4.2.3. Mějme bezkontextovou gramatiku

$$S \to aSX \mid e , \quad X \to XbSb \mid c$$

a odvození

$$S \Rightarrow aSX \Rightarrow aSXbSb \Rightarrow aSXbb \Rightarrow aXbb \Rightarrow acbb .$$

Příslušná levá, resp. pravá odvození téhož slova jsou

$$S \Rightarrow aSX \Rightarrow aX \Rightarrow aXbSb \Rightarrow acbSb \Rightarrow acbb ,$$

resp.

$$S \Rightarrow aSX \Rightarrow aSXbSb \Rightarrow aSXbb \Rightarrow aScbb \Rightarrow acbb .$$

Názorný obraz struktury odvození řetězu podle bezkontextové gramatiky dává tzv. derivační strom. Základní představu o tomto pojmu získáme už z jednoduchého příkladu.

Příklad 4.2.4. Mějme bezkontextovou gramatiku $\mathscr{G}$:

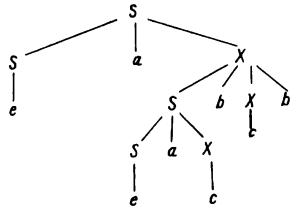
$$S \to SaX \mid e ,$$
$$X \to SbXb \mid c .$$

Řetěz *aacbcb* patří do $L(\mathscr{G})$, neboť je jej možné odvodit levou derivací

$$S \overset{1}{\Rightarrow} SaX \overset{2}{\Rightarrow} aX \overset{3}{\Rightarrow} aSbXb \overset{1}{\Rightarrow} aSaXbXb \overset{2}{\Rightarrow}$$
$$\overset{2}{\Rightarrow} aaXbXb \overset{4}{\Rightarrow} aacbXb \overset{4}{\Rightarrow} aacbcb .$$

Této derivaci odpovídá derivační strom zachycený na obr. 35. Z obrázku je patrné, jakým způsobem se derivační strom vytváří. Kořen stromu je ohodnocen počátečním neterminálem. Hrany vedoucí z vrcholu popsaného neterminálem Y vedou do vrcholů, které jsou ohodnoceny symboly, které čteny zleva doprava dávají nějaký řetěz α takový, že $Y \to \alpha$ je pravidlo dané

gramatiky. Jestliže z vrcholu ohodnoceného Y vede hrana do nějakého vrcholu ohodnoceného e, potom z onoho vrcholu ohodnoceného Y žádná jiná hrana nevede (reprezentace pravidla

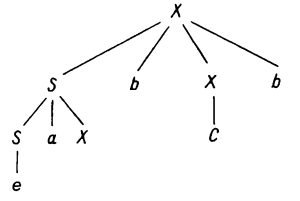


Obr. 35

$Y \to e$). Listy stromu (tj. vrcholy, z nichž nevede žádná hrana) jsou zřejmě ohodnoceny symboly, které čteny zleva doprava dávají generované slovo.

Z uvedeného příkladu je vidět, že v derivačním stromu hraje roli nejen stromové uspořádání dané hranami, ale i uspořádání zleva doprava. V derivačním stromu je kořen ohodnocen počátečním neterminálem a z každého vrcholu ohodnoceného neterminálem vychází nějaká hrana. Odstraníme-li tyto podmínky, dostaneme obecnější pojem stromu popisujícího strukturu řetězu podle bezkontextové gramatiky.

Příklad 4.2.5. Strom na obr. 36 popisuje strukturu řetězu $aXbcb$ podle bezkontextové gramatiky z př. 4.2.4.



Obr. 36

Poznámka 4.2.6. Je ihned zřejmé, že pro libovolnou bezkontextovou gramatiku $\mathscr{G}$, neterminál Y a řetěz ω je $Y \Rightarrow_{\mathscr{G}}^{*} \omega$, právě když existuje strom popisující strukturu ω podle $\mathscr{G}$, jehož kořen je ohodnocen neterminálem Y. Speciálně pro terminální řetěz w je $w \in L(\mathscr{G})$, právě když existuje derivační strom popisující strukturu w podle $\mathscr{G}$.

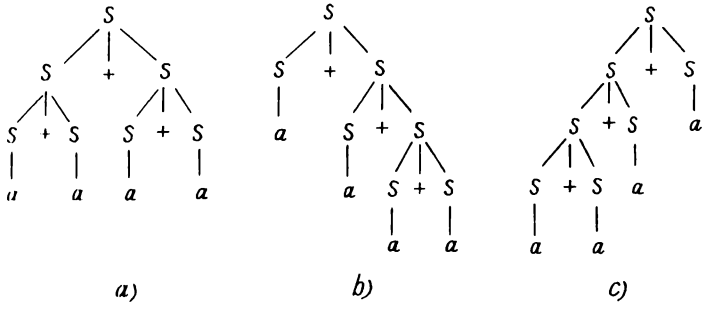
Přesný formální důkaz tvrzení uvedených v poznámce 4.2.6 lze pochopitelně podat až na základě přesné formální definice stromu popisujícího strukturu, resp. derivačního stromu. Jednu z možností, jak tyto pojmy definovat, nalezne čtenář např. v knize [20] spolu s příslušnými důkazy. My se spokojíme s pozorováním obsaženým v následující poznámce, které umožňuje v případě nutnosti převést úvahy o derivačních stromech na úvahy o kanonických derivacích (které jsme přesně definovali).

Poznámka 4.2.7. Každé levé derivaci libovolného terminálního řetězu podle bezkontextové gramatiky odpovídá jediný derivační strom a naopak každému derivačnímu stromu odpovídá jediná levá derivace. Podobně existuje vzájemně jednoznačná korespondence mezi derivačními stromy a pravými derivacemi.

Není obtížné najít bezkontextovou gramatiku, podle níž lze jedno slovo odvodit několika různými levými derivacemi, tzn. lze je popsat několika různými derivačními stromy. Jedná se o tzv. nejednoznačné gramatiky.

Definice 4.2.8. Bezkontextová gramatika $\mathscr{G}$ se nazývá *nejednoznačná* (*víceznačná*), jestliže existuje slovo z jazyka $L(\mathscr{G})$, které má alespoň dvě různé levé derivace podle $\mathscr{G}$. Jinak se $\mathscr{G}$ nazývá *jednoznačná*.

Příklad 4.2.9. 1. Gramatika $S \to S + S \mid a$ (terminální symboly jsou a, $+$) je nejednoznačná, protože např. k řetězu $a + a + a + a$ existuje několik různých derivačních stromů (a tedy také levých



Obr. 37

derivací), jak ukazuje obr. 37. (Na obrázku nejsou zachyceny všechny možné derivační stromy.)

Nejednoznačnost gramatiky způsobuje zpravidla praktické potíže. Nejednoznačná gramatika umožňuje provést rozbor řetězu několika různými způsoby a pokud tento rozbor např. slouží jako podklad pro hledání významu řetězu, vzniká problém, který z významů zvolit. V našem jednoduchém příkladě můžeme slova generovaná gramatikou chápat jako součet několika členů výrazu. Různé rozbory slova $a + a + a + a$ pak vedou k různým způsobům vyhodnocování příslušného součtu. Strom z obr. 37a) by odpovídal sečtení prvních dvou a druhých dvou členů a pak sečtení obou mezivýsledků, obr. 37b) odpovídá vyhodnocování součtů odprava, obr. 37c) vyhodnocování odleva.

Této nepříjemnosti se v tomto případě vyhneme tak, že najdeme ekvivalentní jednoznačnou gramatiku. Příkladem jsou gramatiky $S \to a + S \mid a$ a $S \to S + a \mid a$.

2. Gramatika

$$\begin{aligned} S &\to S_1 \mid S_2 , \\ S_1 &\to AC , \\ S_2 &\to aS_2d \mid aBd , \\ A &\to aAb \mid ab , \\ B &\to bBc \mid bc , \\ C &\to cCd \mid cd \end{aligned}$$

generuje jazyk

$$L = \{a^i b^j c^k d^l ;\ i, j, k, l \geqq 1 \quad \text{a} \quad [(i = j \,\&\, k = l) \text{ nebo } (i = l \,\&\, j = k)]\} .$$

I tato gramatika je nejednoznačná, neboť každý řetěz

$$a^n b^n c^n d^n \quad (n \geqq 1)$$

je možné získat dvojím různým levým odvozením. V tomto případě však není možné najít ekvivalentní jednoznačnou bezkontextovou gramatiku (viz např. [17]). Jazyk L je příkladem tzv. (podstatně) nejednoznačného jazyka.

4.3. Principy analýzy shora

Jedna ze základních fází činnosti moderních překladačů programovacích jazyků je syntaktická analýza. *Syntaktickou analýzu bezkontextových jazyků* můžeme v naší terminologii charakterizovat jako hledání derivačního stromu příslušného k danému slovu a jisté pevně zvolené bezkontextové gramatice. Problematika syntaktické analýzy je dnes postavena na solidních teoretických základech. Celou kap. 5 budeme věnovat exkurzi do této oblasti. Nyní půjde jenom o úvod do tematiky. Probereme jednoduchý příklad a ukážeme, v čem spočívá podstata tzv. analýzy shora. Zároveň nám tento příklad poslouží jako motivace pojmu zásobníkového automatu, definovaného dále.

Vyjděme z (jednoznačné) bezkontextové gramatiky $\mathscr{G}$:

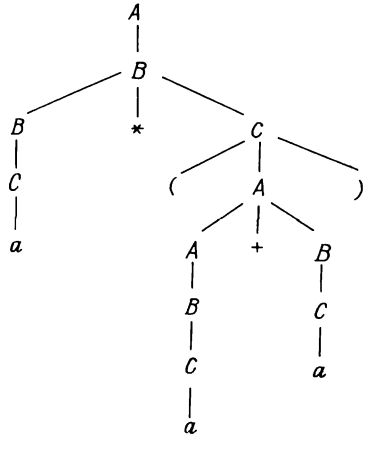
(1) $A \to A + B$, (4) $B \to C$,

(2) $A \to B$, (5) $C \to (A)$,

(3) $B \to B * C$, (6) $C \to a$.

Neterminály této gramatiky jsou A, B, C, počáteční neterminál je A. Terminálními symboly jsou a, $+$, $*$,), (.

Cílem syntaktické analýzy je rozhodnout o libovolném slově w, zda patří do $L(\mathscr{G})$, a jestliže ano, tak sestrojit derivační strom w



Obr. 38

podle $\mathcal{G}$. U jednoznačné gramatiky je pak tento výsledek určen jednoznačně.

Tak např. v našem příkladu slovo $a * (a + a)$ patří do $L(\mathcal{G})$ a příslušný derivační strom je zachycen na obr. 38.

Tento strom by měl být výsledkem analýzy řetězu $a * (a + a)$. Jak víme, jsou levé derivace ve vzájemně jednoznačné korespondenci s derivačními stromy, a proto jako výsledek syntaktické analýzy můžeme stejně dobře požadovat příslušnou levou derivaci, v tomto případě

$$(*) \qquad \begin{aligned} A &\Rightarrow^2 B \Rightarrow^3 B * C \Rightarrow^4 C * C \Rightarrow^6 a * C \Rightarrow^5 a * (A) \Rightarrow^1 \\ &\Rightarrow^1 a * (A + B) \Rightarrow^2 a * (B + B) \Rightarrow^4 a * (C + B) \Rightarrow^6 \\ &\Rightarrow^6 a * (a + B) \Rightarrow^4 a * (a + C) \Rightarrow^6 a * (a + a) . \end{aligned}$$

Nad každou šipkou je v derivaci udáno číslo pravidla, které se v příslušném kroku uplatní. Jelikož se v každém kroku přepisuje nejlevější neterminál, je levá derivace jednoznačně určena už příslušnou posloupností čísel pravidel, v našem případě

$$(**) \qquad 2, 3, 4, 6, 5, 1, 2, 4, 6, 4, 6 .$$

Taková posloupnost je nejstručnější užívanou formou zápisu výsledku syntaktické analýzy.

Metoda syntaktické analýzy shora, kterou se nyní budeme zabývat, není ve své podstatě nic jiného než vytváření levé derivace řetězu spojené s kontrolou, zda počáteční úsek slova, které bylo zatím odvozeno, souhlasí s počátečním úsekem analyzovaného slova. Rozeberme tento postup trochu podrobněji a sledujme, jaké prostředky při něm budeme potřebovat.

Během levé derivace vzniká v každém kroku slovo, které je tvaru $u\eta$, kde u je řetěz terminálů (hotová část slova) a η (nehotová část slova) je řetěz, který, pokud není prázdný, začíná nějakým neterminálem, řekněme X. Jestliže $u \neq e$, máme v okamžiku, kdy dospějeme k $u\eta$, určitou kontrolu správnosti předchozí části derivace, u musí být počátečním úsekem slova w, které se snažíme odvodit, tj. musí být $w = uv$ pro nějaké slovo v. Pokud tomu tak není, je zřejmé, že odvození nevede k cíli. Pokud je, má smysl pokračovat v derivaci a v dalším průběhu už stačí

ověřit, zda slovo v lze odvodit z řetězu η. Do dalšího průběhu analýzy tedy postačuje informace daná dvojicí (v, η). O krok dále to bude nějaká dvojice $(\bar{v}, \bar{\eta})$. Způsob přechodu od (v, η) k $(\bar{v}, \bar{\eta})$ je jednoduchý:

1. Přepíše se neterminál X na levém konci řetězu η, tj. symbol X se nahradí nějakým řetězem ω takovým, že $X \to \omega$ je pravidlem gramatiky $\mathcal{G}$ (aplikace pravidla $X \to e$ pak znamená, že z η se odebere první symbol). Vzniklý řetěz označme $\tilde{\eta}$.

2. Pokud slovo $\tilde{\eta}$ vzniklé z řetězu η podle kroku 1 začíná neterminálem, je $\bar{\eta} = \tilde{\eta}$ a $\bar{v} = v$.

3. Pokud slovo $\tilde{\eta}$ vzniklé z η podle bodu 1 začíná terminálním úsekem $\hat{v}$ (bezprostředně za ním už v $\hat{\eta}$ nenásleduje terminál), potom porovnáme $\hat{v}$ se začátkem v. Jestliže se shodují, potom $\bar{v}$ a η jsou řetězy takové, že $v = \hat{v}\bar{v}$ a $\eta = \hat{v}\bar{\eta}$. Jestliže se slovo $\hat{v}$ neshoduje se začátkem slova v, je tím prokázáno, že zvolená derivace nevede k cíli.

Shrňme operace, které se provádějí při přechodu od dvojice (v, η) k dvojici $(\bar{v}, \bar{\eta})$:

Z řetězu v se odebere určitý počet (popřípadě nulový) počátečních symbolů.

První symbol slova η se nahradí jedním z konečně mnoha různých řetězů připadajících v úvahu a poté se popřípadě z takto vzniklého slova odebere několik prvních (terminálních) symbolů.

Předchozí rozbor vede k volbě datových typů, které při analýze použijeme. Analyzované slovo bude načítáno pomocí vstupního zařízení, které umožňuje kdykoliv přečíst další symbol vstupního slova. Nehotová část slova bude uložena v zásobníkové paměti. *Zásobník*, neboli *LIFO fronta* (z anglického last in, first out), je speciálně organizovaný typ paměti, který umožňuje uchovávat libovolnou posloupnost symbolů z jisté abecedy. Přístup k uchovávané posloupnosti je pouze na jednom z jejích konců. Nejběžněji užívaná terminologie vychází z představy, že zásobník je uspořádán svisle a přístup k zásobníku je pouze na jeho horním konci (vrcholu), tzn. je možné číst pouze symbol na vrcholu zásobníku a buď vrchní symbol ze zásobníku ubrat, nebo naopak přidat nový symbol na vrchol zásobníku.

Průběh analýzy slova $a * (a + a)$ z našeho příkladu zachycuje

tab. 5. Pro úsporu místa je v tabulce obsah zásobníku znázorněn vodorovně, tj. řetězem, levý konec řetězu ztotožňujeme s vrcholem zásobníku, pravý konec se dnem zásobníku.

Obsah zásobníku	Čtený vstupní symbol	Prováděná operace	Číslo použitého pravidla gramatiky
A	–	první symbol zásobníku se nahradí pravou stranou pravidla č. 2, tj. $A \rightarrow B$	2
B	–	první symbol se nahradí pravou stranou pravidla $B \rightarrow B * C$	3
$B * C$	–	první symbol se nahradí pravou stranou pravidla $B \rightarrow C$	4
$C * C$	–	první symbol se nahradí podle pravidla $C \rightarrow a$	6
$a * C$	a	první symbol je terminální, a proto se srovná s čteným symbolem vstupního slova. V případě souhlasu (jako zde) se první symbol zásobníku odstraní a čte se následující vstupní symbol. V případě nesouhlasu by se přerušil výpočet	–
$* C$	$*$	obdobně jako v minulém kroku	–
C	–	C se přepíše na (A)	5
(A)	(	odstraní se (ze zásobníku a na vstup se připraví další symbol vstupního slova	–
$A)$	–	A se přepíše na $A + B$	1
$A + B)$	–	A se přepíše na B	2
$B + B)$	–	B se přepíše na C	4
$C + B)$	–	C se přepíše na a	6
$a + B)$	a	odstraní se a a přejde na další vstupní symbol	–
$+B)$	$+$	obdobně jako v minulém kroku	–
$B)$	–	B se přepíše na C	4
$C)$	–	C se přepíše na a	6
$a)$	a	odstraní se a a přejde na další vstupní symbol	–
$)$	$)$	obdobně	–
–	slovo přečteno		

Tab. 5

Sledujeme-li průběh symbolů, které se během výpočtu objevují v zásobníku, a konfrontujeme-li tento průběh s derivačním stromem na obr. 38, vidíme, že výpočet odpovídá konstrukci stromu směrem shora. Odtud také pochází název příslušné metody syntaktické analýzy.

Dosud jsme zcela ignorovali podstatnou otázku, totiž jakým způsobem se určuje, podle kterého pravidla se má přepsat vrchní (první) symbol v zásobníku. To je skutečně důležitý problém, jehož řešení vlastně bude věnována celá kap. 5. Zatím se mu vyhneme tak, že budeme předpokládat existenci jakéhosi dodatečného zdroje informací o tom, které pravidlo v každém okamžiku použít. Jak si tento zdroj informace opatřit, budeme zkoumat v dalším textu, teď jenom rozebereme, jaký vliv může případná nespolehlivost tohoto zdroje mít na úspěšnost analýzy.

Závěry jsou jasné: Každé slovo patřící do $L(\mathscr{G})$ je při správné funkci zdroje informací úspěšně analyzováno. Ani vadná funkce dodatečného zdroje informací nemůže vést k přijetí slova nepatřícího do $L(\mathscr{G})$. Při libovolných dodatečných informacích je totiž zahájen výpočet odpovídající nějaké derivaci. Jestliže je to derivace vedoucí k jinému slovu, než je analyzované slovo, pak se tato skutečnost v průběhu výpočtu zjistí (je zjištěna neshoda mezi vstupem a vrcholem zásobníku nebo je po přečtení celého vstupního slova neprázdný zásobník nebo je zásobník vyprázdněn před přečtením celého vstupního slova). Jestliže je to derivace nevedoucí k žádnému terminálnímu řetězu, nemůže dojít k vyprázdnění zásobníku.

Situace, kdy celé vstupní slovo je přečteno a zásobník je prázdný, je tedy dostatečnou zárukou, že zdroj dodatečných informací fungoval správně a že analyzované slovo patří do jazyka $L(\mathscr{G})$.

Úvahy, které jsme provedli, vyústí v čl. 4.4 v zavedení několika pojmů. Prostředky analýzy, které jsme použili, se přesně odrazí v definici zásobníkového automatu a jazyka, který tento automat rozpoznává. Popsaná metoda analýzy najde své formální vyjádření v důkazu věty, že každý bezkontextový jazyk je rozpoznatelný zásobníkovým automatem. Odsunutí otázky, jak rea-

lizovat zdroj dodatečných informací a soustředění se na obecný mechanismus analýzy vede k tomu, že zásobníkový automat je definován jako nedeterministický.

4.4. Zásobníkové automaty

Definice 4.4.1. *Zásobníkovým automatem* nazýváme sedmici

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

kde Q je neprázdná konečná množina stavů, Σ je neprázdná konečná vstupní abeceda, Γ je neprázdná konečná zásobníková abeceda, $q_0 \in Q$ je počáteční stav, $Z_0 \in \Gamma$ je počáteční zásobníkový symbol (na začátku výpočtu bude zásobník obsahovat pouze symbol Z_0), $F \subseteq Q$ je množina koncových stavů, δ je zobrazení $Q \times (\Sigma \cup \{e\}) \times \Gamma$ do množiny všech konečných podmnožin [redacted] $\mathcal{P}(Q \times \Gamma^*)$

Z definice je patrné, že zásobníkový automat je obecně nedeterministický. Věnujme pozornost způsobu, jakým budeme interpretovat zobrazení δ:

$$\delta(q, a, X) = \{(q_1, \eta_1), (q_2, \eta_2), \ldots, (q_m, \eta_m)\},$$

kde

$$q, q_1, \ldots, q_m \in Q, \; a \in \Sigma, \; X \in \Gamma, \; \eta_1, \ldots, \eta_m \in \Gamma^*,$$

znamená: jestliže je automat $\mathcal{M}$ ve stavu q, čte vstupní symbol a a na vrcholu zásobníku je symbol X, pak může přejít do stavu q_i a symbol X nahradit řetězem η_i (pro nějaké i, $1 \leqq i \leqq m$). V takovém případě budeme také někdy říkat, že zásobníkový automat $\mathcal{M}$ vykonal *instrukci* $(q, a, X) \to (q_i, \eta_i)$.

K tomu ještě dvě poznámky:

1. V souladu s dohodou z čl. 4.3 budeme předpokládat, že první symbol řetězu η_i bude v zásobníku uložen nejvýš, druhý zleva bude jako druhý shora atd.

2. Všimněte si dobře, že interpretace instrukcí předpokládá, že vrchní symbol zásobníku se pokaždé nahrazuje. Chceme-li tedy např. nechat zásobník beze změny, je tak třeba učinit pravidlem, které do zásobníku vkládá stejný symbol, jako se právě odebral.

Interpretace pravidla

$$(*) \qquad \delta(q, e, X) = \{(q_1, \eta_1), \ldots, (q_m, \eta_m)\}$$

je tato: jestliže automat je ve stavu q a vrchní symbol v zásobníku je X, potom automat, aniž přijme na vstup další symbol vstupního slova (tj. aniž čte vstupní symbol a posouvá hlavu na vstupní pásce) přejde do stavu q_i a symbol X nahradí slovem η_i (pro nějaké i, $1 \leqq i \leqq m$). Tzn. zpracovává-li zásobníkový automat vstupní slovo $a_1 \ldots a_n$, má-li v určitém okamžiku na vstupu i-tý symbol slova a jestliže v tomto okamžiku provádí některou z instrukcí $(*)$, pak i v následujícím okamžiku má na vstupu i-tý symbol slova. V takovém případě říkáme, že *automat provedl ε-krok*.

Důležitým detailem v definici 4.4.1 je i skutečnost, že zobrazení δ může nabývat hodnoty $\emptyset$ (prázdná množina). Tyto hodnoty budeme v zadání δ prostě vynechávat a budeme říkat, že pro příslušné argumenty není δ definováno. V jistých situacích tedy nemůže automat pokračovat ve výpočtu.

Předchozí poznámky shrneme v následující definici.

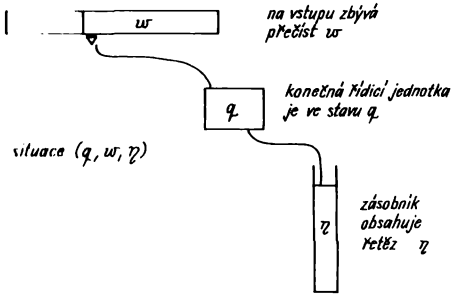
Definice 4.4.2. *Situací zásobníkového automatu*

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

nazveme trojici (q, w, η), kde

$$q \in Q, \quad w \in \Sigma^*, \quad \eta \in \Gamma^*$$

(viz obr. 39).



Obr. 39

Řekneme, že *situace* $(q, aw, Y\eta)$ *automatu* $\mathscr{M}$ *vede bezprostředně k situaci* $(q', w, v\eta)$, symbolicky

$$(q, aw, Y\eta) \vdash_{\mathscr{M}} (q', w, v\eta),$$

pro

$$a \in \Sigma \cup \{e\},$$

jestliže

$$(q', v) \in \delta(q, a, Y).$$

Jestliže E a E' jsou dvě situace automatu $\mathscr{M}$, řekneme, že situace E vede k situaci E' (symbolicky $E \vdash^*_{\mathscr{M}} E'$), právě když existují situace $E_1, \ldots, E_n$ automatu $\mathscr{M}$ tak, že $E = E_1$, $E_n = E'$ a $E_i \vdash_{\mathscr{M}} E_{i+1}$ (pro všechna i, $1 \leqq i < n$), nebo když $E = E'$.

Situace zásobníkového automatu tedy udává:

1. stav, ve kterém se automat nachází;
2. úsek vstupního slova, který má automat ještě projít;
3. obsah zásobníku.

Poznámka 4.4.3. Z definice 4.4.2 také plyne, že situace, ve které je zásobník prázdný, už k žádné další situaci nevede.

Pro přijímání slova zásobníkovým automatem jsou zavedeny dvě definice:

a) Analogicky jako u konečných automatů je *slovo přijato zásobníkovým automatem* tehdy, jestliže po zpracování posledního symbolu vstupního slova se automat ocitne v některém koncovém stavu – tzv. *přijímání koncovým stavem.*

b) Podobně jako u metody analýzy, kterou jsme probrali v předchozím článku, je *slovo přijato zásobníkovým automatem* právě tehdy, když v okamžiku, kdy je celé zpracováno, má automat prázdný zásobník – tzv. *přijímání prázdným zásobníkem.*

Definice 4.4.4. Nechť $\mathscr{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ je zásobníkový automat. Definujme

1. *jazyk* $L(\mathscr{M})$ *rozpoznávaný koncovým stavem*:

$$L(\mathscr{M}) = \{w; (q_0, w, Z_0) \vdash^*_{\mathscr{M}} (q, e, \eta)$$
$$\text{pro nějaké } q \in F \text{ a nějaké } \eta \in \Gamma^*\};$$

2. *jazyk* $N(\mathscr{M})$ *rozpoznávaný prázdným zásobníkem*:

$$N(\mathscr{M}) = \{w; (q_0, w, Z_0) \vdash_{\mathscr{M}}^{*} (q, e, e) \text{ pro nějaké } q \in Q\}.$$

Navrhujeme-li zásobníkový automat pro rozpoznávání prázdným zásobníkem, je množina F koncových stavů nepodstatná. Proto v tom případě obvykle klademe $F = \emptyset$.

Příklad 4.4.5. Sestrojme zásobníkový automat, který rozpoznává jazyk $L = \{w(w)^R; w \in \{0, 1\}^*\}$ prázdným zásobníkem. Hledaný automat

$$\mathscr{M} = (\{p, q\}, \{0, 1\}, \{A, B, C\}, \delta, p, A, \emptyset)$$

má přechodovou funkci δ definovánu takto:

1. $\delta(p, 0, A) = \{(p, BA)\}$,
2. $\delta(p, 1, A) = \{(p, CA)\}$,
3. $\delta(p, 0, B) = \{(p, BB), (q, e)\}$,
4. $\delta(p, 0, C) = \{(p, BC)\}$,
5. $\delta(p, 1, B) = \{(p, CB)\}$,
6. $\delta(p, 1, C) = \{(p, CC), (q, e)\}$,
7. $\delta(q, 0, B) = \{(q, e)\}$,
8. $\delta(q, 1, C) = \{(q, e)\}$,
9. $\delta(p, e, A) = \{(q, e)\}$,
10. $\delta(q, e, A) = \{(q, e)\}$.

Popišme, na jakém principu $\mathscr{M}$ rozpoznává jazyk L. V první části výpočtu automat ukládá do zásobníku zápis přečteného úseku slova. Symbol 0 kóduje zásobníkovým symbolem B, symbol 1 kóduje symbolem C. Dokud se stroj „domnívá“, že čte první polovinu slova, ukládá do zásobníku kódovaný zápis a zůstává ve stavu p. Tomu odpovídají pravidla 1 až 6, přesněji řečeno u pravidel 3 a 6 jejich první možnost. V okamžiku, kdy automat „dojde k názoru“, že se ocitl přesně v polovině slova, přejde do stavu q. Je-li ovšem automat skutečně přesně za polovinou slova a patří-li čtené slovo do jazyka L, potom se symbol, který je právě na vstupu, musí shodovat se symbolem, který byl čten v minulém okamžiku. Proto je přechod do stavu q umožněn

u pravidel 3 a 6 a nikoliv u 4 a 5. Od okamžiku, kdy $\mathscr{M}$ přechází do stavu q, vybírá ze zásobníku symboly a srovnává je se vstupními symboly. Tím kontroluje (pravidla 7, 8), zda druhá část slova je zrcadlovým obrazem první části. Není-li tomu tak, potom se $\mathscr{M}$ dostane do situace, pro kterou příslušná hodnota funkce δ není definována. Jestliže dané slovo patří do L a jestliže automat správně „odhadl" prostředek slova, pak v okamžiku, kdy automat zpracuje poslední symbol vstupního slova, zbývá v zásobníku pouze symbol A. Lze pak aplikovat pravidlo 10 a slovo je přijato. Pravidlo 9 umožňuje přijmout prázdné slovo.

Pokud automat přejde do stavu q jindy než uprostřed slova, pak buď dojde k vyprázdnění zásobníku ještě před koncem slova, nebo po přečtení celého slova obsahuje zásobník nad symbolem A ještě další symboly.

Pro každé slovo z L tedy zřejmě existuje přijímající výpočet a obráceně žádné slovo nepatřící do L není automatem $\mathscr{M}$ přijato.

Ukážeme nyní, že oba způsoby reprezentace jazyka zásobníkovým automatem jsou rovnocenné.

Lemma 4.4.6. *Nechť L je libovolný jazyk. Pak $L = N(\mathscr{M}_1)$ pro nějaký zásobníkový automat $\mathscr{M}_1$, právě když $L = L(\mathscr{M}_2)$ pro nějaký zásobníkový automat $\mathscr{M}_2$.*

Důkaz. a) Předpokládejme, že $L = L(\mathscr{M}_2)$ pro nějaký zásobníkový automat $\mathscr{M}_2 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$. Sestrojíme $\mathscr{M}_1$ tak, že $N(\mathscr{M}_1) = L(\mathscr{M}_2)$. Definujme

$$\mathscr{M}_1 = (Q \cup \{q_e, p\}, \Sigma, \Gamma \cup \{R\}, \delta', p, R, \emptyset),$$

kde δ' je definováno takto:

1. $\delta'(p, e, R) = \{(q_0, Z_0 R)\}$,

2. pro všechna

$q, \hat{q} \in Q, a \in \Sigma \cup \{e\}, D \in \Gamma,$

je

$(\hat{q}, \eta) \in \delta'(q, a, D)$, právě když $(\hat{q}, \eta) \in \delta(q, a, D)$,

3. $(q_e, e) \in \delta'(q, e, D)$

pro všechna $q \in F, D \in \Gamma \cup \{R\}$,

4. $\delta'(q_e, e, D) = \{(q_e, e)\}$

pro všechna $D \in \Gamma \cup \{R\}$.

Automat $\mathcal{M}_1$ pracuje tak, že nejprve pomocí pravidla 1 přejde do počáteční situace automatu $\mathcal{M}_2$ upravené tím způsobem, že pod počátečním symbolem automatu $\mathcal{M}_2$ je nyní v zásobníku uložen nový počáteční smybol R. Potom pomocí pravidel typu 2 simuluje $\mathcal{M}_1$ činnost automatu $\mathcal{M}_2$. Jakmile se $\mathcal{M}_2$ dostane během výpočtu do koncového stavu, potom $\mathcal{M}_1$ může v odpovídajícím místě výpočtu přejít pomocí pravidla typu 3 do stavu q_e. V tomto stavu vyprázdní pomocí pravidel 4 celý zásobník.

Ještě vysvětlíme, proč má $\mathcal{M}_1$ uložen pod symbolem Z_0 nový počáteční symbol R. Automat $\mathcal{M}_2$ přijímá koncovým stavem. Jestliže má tedy na konci zpracování nějakého slova $w \in L$ prázdný zásobník, je to pro přijetí slova nepodstatné. Kdyby se však při simulaci výpočtu automatu $\mathcal{M}_2$ automatem $\mathcal{M}_1$ vyprázdnil na konci výpočtu zásobník, znamenalo by to, že je w přijato. Tomu zabrání právě symbol R, který se ze zásobníku může odstranit až po skončení simulační části výpočtu.

b) Předpokládejme, že $L = N(\mathcal{M}_1)$ pro zásobníkový automat

$$\mathcal{M}_1 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset).$$

Sestrojíme $\mathcal{M}_2$ tak, že

$$L(\mathcal{M}_2) = N(\mathcal{M}_1).$$

Položme

$$\mathcal{M}_2 = (Q \cup \{p, q_f\}, \Sigma, \Gamma \cup \{R\}, \delta', p, R, \{q_f\}),$$

kde δ' je definováno takto:

1. $\delta'(p, e, R) = \{(q_0, Z_0 R)\}$,

2. $\delta'(q, a, Z) = \delta(q, a, Z)$

pro všechna $q \in Q, a \in \Sigma \cup \{e\}, Z \in \Gamma$,

3. $\delta'(q, e, R) = (q_f, e)$ pro všechna $q \in Q$.

Automat $\mathscr{M}_2$ na základě pravidla 1 přejde do počáteční situace automatu $\mathscr{M}_1$ modifikované tak, že pod symbolem Z_0 je ještě v zásobníku symbol R. Pravidla typu 2 simulují výpočet automatu $\mathscr{M}_1$. Jestliže $\mathscr{M}_1$ během výpočtu vyprázdní celý zásobník, potom $\mathscr{M}_2$ v odpovídajícím místě simulace výpočtu $\mathscr{M}_1$ také vyprázdní celý zásobník až na symbol R. Jakmile se objeví symbol R, převede pravidlo typu 3 automat $\mathscr{M}_2$ do koncového stavu q_f.

U deterministické verze zásobníkových automatů lemma neplatí, jak ukážeme.

Definice 4.4.7. *Zásobníkový automat*

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

se nazývá *deterministický*, právě když jsou splněny tyto dvě podmínky:

1. Jestliže $\delta(q, e, Z) \neq \emptyset$, potom $\delta(q, a, Z) = \emptyset$ pro libovolné $a \in \Sigma$ $(q \in Q, Z \in \Gamma)$.

2. Pro libovolné $q \in Q$, $Z \in \Gamma$ a $a \in \Sigma \cup \{e\}$ obsahuje $\delta(q, a, Z)$ nejvýše jeden prvek.

Podmínka 1 vylučuje nedeterminismus, který by se projevil tak, že v určité situaci by bylo možné pokračovat buď e-krokem, nebo krokem spojeným se čtením následujícího vstupního symbolu.

Lemma 4.4.8. 1. *Ke každému deterministickému zásobníkovému automatu $\mathscr{M}_1$ lze sestrojit deterministický zásobníkový automat $\mathscr{M}_2$ takový, že $N(\mathscr{M}_1) = L(\mathscr{M}_2)$.*

2. *Existuje regulární jazyk, který není rozpoznatelný žádným deterministickým zásobníkovým automatem pomocí prázdného zásobníku.*

Důkaz. Část 1 se dokáže stejně jako část b) v důkazu 4.4.6. Jestliže jako výchozí automat $\mathscr{M}_1$ v tomto důkazu zvolíme de-

deterministický zásobníkový automat, je i výsledný automat $\mathscr{M}_2$ deterministický, jak okamžitě plyne z konstrukce.

Část 2 plyne z pozorování, že po vyprázdnění zásobníku nemůže automat pokračovat ve výpočtu (viz 4.4.3). Jestliže tedy pro libovolné v a libovolný deterministický zásobníkový automat $\mathscr{M}$ je $v \in N(\mathscr{M})$, je pro každé $u \neq e$ nutně $vu \notin N(\mathscr{M})$. Z determinismu $\mathscr{M}$ totiž plyne, že po přečtení počátečního úseku v slova vu se vyprázdní zásobník (protože v je přijato prázdným zásobníkem) a slovo vu není dočteno do konce, takže není přijato. Proto např. konečný jazyk $\{a, aa\}$ není deterministicky rozpoznatelný prázdným zásobníkem.

Poznámka 4.4.9. 1. Na rozpoznávání jazyka deterministickým konečným automatem lze pohlížet jako na speciální případ rozpoznávání deterministickým zásobníkovým automatem pomocí koncových stavů (zásobník není využíván). Proto z lemmatu 4.4.8 plyne, že v případě deterministických zásobníkových automatů je rozpoznávání koncovými stavy silnější prostředek než rozpoznávání prázdným zásobníkem.

2. Část 2 lemmatu 4.4.8 lze doplnit příkladem neregulárního jazyka $L = \{0^n 1^n; n \geqq 1\}$ (viz 1.3.6), který je rozpoznáván pomocí prázdného zásobníku deterministickým zásobníkovým automatem

$$\mathscr{M} = (\{q_1, q_2\}, \{0, 1\}, \{Z, A\}, \delta, q_1, Z, \emptyset),$$

jehož přechodová funkce δ je definována takto:

$$\begin{aligned}
\delta(q_1, 0, Z) &= (q_1, AZ), \\
\delta(q_1, 0, A) &= (q_1, AA), \\
\delta(q_1, 1, A) &= (q_2, e), \\
\delta(q_2, 1, A) &= (q_2, e), \\
\delta(q_2, e, Z) &= (q_2, e).
\end{aligned}$$

Třída regulárních jazyků a třída jazyků rozpoznatelných deterministickým zásobníkovým automatem pomocí prázdného zásobníku jsou tedy nesrovnatelné inkluzí.

Definice 4.4.10. *Deterministickým jazykem* budeme nazývat každý jazyk rozpoznatelný deterministickým zásobníkovým auto-

matem pomocí koncového stavu. *Bezprefixovým deterministickým jazykem* budeme nazývat každý jazyk rozpoznatelný deterministickým zásobníkovým automatem pomocí prázdného zásobníku.

Rozpoznávání prázdným zásobníkem je znevýhodněno tím, že automat nemá informaci o tom, zda vstupní slovo bylo přečteno celé. Dodáním této informace, např. speciálním koncovým znakem, se rozdíl mezi oběma druhy rozpoznávání setře, jak ukazuje následující lemma.

Lemma 4.4.11. *Nechť $L \subseteq \Sigma^*$ je libovolný deterministický jazyk a $\S \notin \Sigma$. Potom $L.\{\S\}$ je bezprefixový deterministický jazyk.*

Důkaz tohoto lemmatu je snadnou modifikací části a) důkazu lemmatu 4.4.6, a proto jej přenecháme čtenáři jako cvičení.

4.5. Zásobníkové automaty a bezkontextové jazyky

V tomto článku ukážeme, že nedeterministické zásobníkové automaty rozpoznávají právě bezkontextové jazyky. Jedna část tohoto tvrzení se získá formalizací postupu analýzy shora, který jsme rozebírali v čl. 4.3.

Věta 4.5.1. *Ke každému bezkontextovému jazyku L existuje nedeterministický zásobníkový automat $\mathscr{M}$ (s jediným stavem) takový, že $L = N(\mathscr{M})$.*

Důkaz. Nechť $L = L(\mathscr{G})$ pro nějakou bezkontextovou gramatiku $\mathscr{G} = (\Pi, \Sigma, S, P)$. Sestrojíme zásobníkový automat

$$\mathscr{M} = (\{q\}, \Sigma, \Pi \cup \Sigma, \delta, q, S, \emptyset),$$

kde δ je definováno takto:

1. $\delta(q, e, D) = \{(q, \eta); D \to \eta \text{ je pravidlo z } P\}$ pro všechna $D \in \Pi$,
2. $\delta(q, a, a) = \{(q, e)\}$ pro každé $a \in \Sigma$.

Dokážeme, že pro takto definovaný automat $\mathscr{M}$ je $N(\mathscr{M}) = L$. V každém kroku libovolného výpočtu automatu $\mathscr{M}$ se buď

(i) nahrazuje vrchní symbol v zásobníku nějakým řetězem (podle pravidel typu 1), nebo

(ii) odebírá vrchní symbol ze zásobníku (podle 2).

Zvolme libovolné slovo $w \in \Sigma^*$ a uvažujme libovolný konečný výpočet (délky r) stroje $\mathscr{M}$ nad vstupním slovem w. Označme u_i řetěz symbolů, které byly od začátku výpočtu až do skončení i-tého kroku odebrány ze zásobníku podle pravidel typu 2 (je zřejmě $u_i \in \Sigma^*$), a η_i obsah zásobníku po skončení i-tého kroku.

Dále položme $\omega_i = u_i\eta_i$. Pro každé i, $0 \leqq i \leqq r$, platí toto:

(a) buď $\omega_i \Rightarrow_{\mathscr{G}} \omega_{i+1}$ a ω_{i+1} vznikne z ω_i přepsáním nejlevějšího neterminálu (to odpovídá pravidlu typu 1),

(b) nebo $\omega_i = \omega_{i+1}$ (to odpovídá pravidlu typu 2).

Výpočet tedy sleduje jistou derivaci podle gramatiky $\mathscr{G}$.

Naopak také kroky levé derivace lze simulovat automatem $\mathscr{M}$:

Nechť $\omega = u\eta$ a $\omega' = u'\eta'$ jsou libovolné řetězy takové, že u i u' jsou terminální řetězy;

každé z η, η' buď začíná neterminálem, nebo je prázdné;

$\omega \Rightarrow_{\mathscr{G}} \omega'$ levým přepsáním.

Potom $(q, v, \eta) \vdash^* (q, e, \eta')$, kde $uv = u'$. (Výpočet začne aplikací pravidla typu 1, popřípadě následovanou aplikací pravidel typu 2.)

Tvrzení, že $L(\mathscr{G}) = N(\mathscr{M})$, plyne ze skutečnosti, že všechna následující tvrzení jsou vzájemně ekvivalentní.

(1) $w \in N(\mathscr{M})$.

(2) $(q, w, S) \vdash^*_{\mathscr{M}} (q, e, e)$.

(3) Existuje výpočet délky r (pro nějaké r) automatu $\mathscr{M}$ nad w, pro který (podle předchozího značení) platí toto:

$$u_0 = e \,\&\, \eta_0 = S, \text{ tj. } \omega_0 = S$$

a

$$u_r = w \,\&\, \eta_r = e, \text{ tj. } \omega_r = w.$$

(4) $S \Rightarrow^*_{\mathscr{G}} w$.

(5) $w \in L(\mathscr{G})$.

Skutečně:

(1) $\Leftrightarrow$ (2) $\Leftrightarrow$ (3) podle definice výpočtu a přijímání slova prázdným zásobníkem.

(3) $\Leftrightarrow$ (4) podle předchozích úvah.

(4) $\Leftrightarrow$ (5) podle definice 3.1.5 jazyka generovaného gramatikou.

Příklad 4.5.2. Jazyk $L = \{a^i b^j c^k;\ i + j = k\}$ generovaný gramatikou

$$A \to aAc \mid B,$$
$$B \to bBc \mid e \quad (A \text{ je počáteční symbol})$$

je rozpoznáván prázdným zásobníkem zásobníkovým automatem

$$\mathcal{M} = (\{q\}, \{a, b, c\}, \{A, B, a, b, c\}, \delta, q, A, \emptyset),$$

kde δ je definováno takto:

$$\delta(q, e, A) = \{(q, aAc), (q, B)\},$$
$$\delta(q, e, B) = \{(q, bBc), (q, e)\},$$
$$\delta(q, a, a) = \delta(q, b, b) = \delta(q, c, c) = \{(q, e)\}.$$

Od zásobníkového automatu je možné přejít k ekvivalentní bezkontextové gramatice zcela analogickým způsobem v případě, že automat má jediný stav. To budeme demonstrovat v důkazu věty 4.5.5. Dosah této skutečnosti vysvitne až ve spojení s následující větou, která ukazuje, že pomocí rozšíření zásobníkové abecedy lze redukovat počet stavů automatu na jeden (v případě rozpoznávání prázdným zásobníkem).

Věta 4.5.3. *Ke každému zásobníkovému automatu $\mathcal{M}$ lze sestrojit zásobníkový automat $\mathcal{M}'$ s jediným stavem takový, že*

$$N(\mathcal{M}) = N(\mathcal{M}').$$

Důkaz. Mějme $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$. Vysvětleme si nejprve v hrubých rysech, na jakém principu $\mathcal{M}'$ pracuje. $\mathcal{M}'$ má zásobníkové symboly tvaru $\langle q, B, q' \rangle$, kde $q, q' \in Q$ a $B \in \Gamma$.

Výpočty $\mathcal{M}'$ probíhají jakoby souběžně s výpočty $\mathcal{M}$. Jestliže se během výpočtu $\mathcal{M}'$ vyskytuje v zásobníku symbol $\langle q, B, q' \rangle$, nese takovou informaci:

1. v odpovídajícím kroku příslušného výpočtu $\mathcal{M}$ je na odpovídajícím místě zásobníku symbol B;
2. až $\mathcal{M}$ dospěje do situace, že bude tento symbol zpracovávat, bude ve stavu q;
3. až $\mathcal{M}$ dospěje do situace, že bude zpracovávat symbol uložený bezprostředně pod tímto B, bude ve stavu q'.

Zajistit správnost informace 1 není problém. K tomu stačí provádět výpočty $\mathscr{M}'$ souběžně s výpočty $\mathscr{M}$. Informace 2 a 3 se týkají i budoucích kroků výpočtů. Proto je bude $\mathscr{M}'$ nedeterministicky „hádat". Musí mít ovšem příležitost později ověřit správnost tohoto „hádání". To se podaří, jestliže při výpočtu pokračuje tak, aby se nedostal do sporu s předpokládaným významem jednotlivých složek zásobníkových symbolů. Speciálně to znamená toto:

Z 2 a 3 okamžitě plyne, že pokud je pod $\langle q, B, q' \rangle$ bezprostředně uložen symbol $\langle \bar{q}, C, \tilde{q} \rangle$, musí být $q' = \bar{q}$. Proto jestliže je do zásobníku vkládán řetěz

$$\langle q_1, A_1, q_1' \rangle \langle q_2, A_2, q_2' \rangle \ldots \langle q_k, A_k, q_k' \rangle ,$$

musí platit, že $q_i' = q_{i+1}$ pro všechna i, $1 \leqq i < k$. Jestliže se tímto řetězem nahrazuje symbol $\langle q, B, q' \rangle$, musí navíc $q' = q_k'$. Zároveň z 2 plyne, že q_1 musí být stav, ve kterém je v okamžiku odpovídajícího výpočtu automat $\mathscr{M}$. (Formálně je to zachyceno v pravidle 3 následující definice funkce δ'.)

Z 2 a 3 také plyne, že jestliže automat $\mathscr{M}$ v určitém okamžiku výpočtu odebírá ze zásobníku symbol B a přitom přechází ze stavu q do q', musí v odpovídajícím kroku výpočtu $\mathscr{M}'$ být na vrcholu zásobníku symbol $\langle q, B, q' \rangle$ (viz bod 2 následující definice funkce δ'). V takovém okamžiku se tedy kontroluje správnost dříve učiněné „předpovědi" o stavu q'.

Nyní podáme přesnou konstrukci $\mathscr{M}'$. Vyjdeme od automatu

$$\mathscr{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset) .$$

Sestrojíme

$$\mathscr{M}' = (\{p\}, \Sigma, \Gamma', \delta', p, R, \emptyset) ,$$

kde

$$\Gamma' = Q \times \Gamma \times Q \cup \{R\}$$

(R je nově přidaný symbol) a přechodová funkce δ' je definována takto:

1. $\delta'(p, e, R) = \{(p, \langle q_0, Z_0, q_1 \rangle); q_1 \in Q\}$;
2. ke každé instrukci

$$(q, a, D) \xrightarrow{\delta} (q', e)$$

automatu $\mathscr{M}$ bude mít $\mathscr{M}'$ instrukci

$$(p, a, \langle q, D, q' \rangle) \xrightarrow{\delta'} (p, e)$$

pro libovolná $q, q' \in Q$, $a \in \Sigma \cup \{e\}$, $D \in \Gamma$;

3. ke každé instrukci

$$(q, a, D) \xrightarrow{\delta} (q', B_1 \dots B_k)$$

(pro $q, q' \in Q$, $a \in \Sigma \cup \{e\}$, $B_1, \dots, B_k \in \Gamma$) bude mít $\mathscr{M}'$ všechny instrukce typu

$$(p, a, \langle q, D, \tilde{q} \rangle) \rightarrow$$
$$\rightarrow (p, \langle q', B_1, q_1 \rangle \langle q_1, B_2, q_2 \rangle \dots \langle q_{k-1}, B_k, \tilde{q} \rangle,$$

kde $q_1, \dots, q_{k-1}$, $\tilde{q} \in Q$ jsou libovolné.

Jiné instrukce mít $\mathscr{M}'$ nebude.

Zbývá dokázat, že $N(\mathscr{M}) = N(\mathscr{M}')$. Označme pro každou instrukci I automatu $\mathscr{M}$ symbolem $h(I)$ množinu instrukcí automatu $\mathscr{M}'$ definovaných podle bodů 2 a 3. [Pro instrukce definované podle 2 je tedy $h(I)$ jednobodová množina.] Dále pro každou situaci E automatu $\mathscr{M}$ definujme množinu $H(E)$ situací automatu $\mathscr{M}'$ takto:

$$(*) \qquad H((q, u, A_1 \dots A_l)) =$$
$$= \{(p, u, \langle q, A_1, q_1 \rangle \langle q_1, A_2, q_2 \rangle \dots$$
$$\dots \langle q_{l-1}, A_l, q_l \rangle); q_1, \dots, q_l \in Q\}, \quad l \geqq 1,$$

$$(**) \qquad H((q, u, e)) = \{(p, u, e)\}.$$

Pro každé E je tedy $H(E) \neq \emptyset$.

Přímo z definice funkce δ' plyne, že

a) Pro každou počáteční situaci (q_0, w, Z_0) automatu $\mathscr{M}$ je $H((q_0, w, Z_0))$ přesně množina situací, do kterých se $\mathscr{M}'$ může dostat po prvním kroku výpočtu, tj. po provedení některé instrukce definované v bodu 1.

b) Jestliže $E_1 \vdash^I_{\mathscr{M}} E_2$, potom ke každému $E_2' \in H(E_2)$ existuje $I' \in h(I)$ a $E_1' \in H(E_1)$ tak, že $E_1' \vdash^{I'}_{\mathscr{M}'} E_2'$.

c) Jestliže $E_1' \vdash^{I'}_{\mathscr{M}'} E_2'$ a jestliže $E_1' \in H(E_1)$ pro nějaké E_1, potom $I' \in h(I)$ pro nějaké I a $E_2' \in H(E_2)$ pro nějaké E_2 tak, že $E_1 \vdash^I_{\mathscr{M}} E_2$.

Z b) a c) plyne, že $E_1 \vdash^*_{\mathscr{M}} E_2$, právě když $E'_1 \vdash^*_{\mathscr{M}'} E'_2$ pro nějaké $E'_1 \in H(E_1)$ a $E'_2 \in H(E_2)$. Je tedy také (pro nějaké $q \in Q$)

$$(q_0, w, Z_0) \vdash^*_{\mathscr{M}} (q, e, e),$$

právě když $E'_1 \vdash^*_{\mathscr{M}'} E'_2$ pro nějaké

$$E'_1 \in H((q_0, w, Z_0)) \text{ a } E'_2 \in H((q, e, e)).$$

To podle a) a (**) nastane právě tehdy, když

$$(p, w, R) \vdash^*_{\mathscr{M}'} (p, e, e).$$

Tedy $w \in N(\mathscr{M})$, právě když $w \in N(\mathscr{M}')$.

Příklad 4.5.4. K zásobníkovému automatu

$$\mathscr{M} = (\{p, q\}, \{0, 1\}, \{A, B\}, \delta, p, A, \emptyset),$$

kde δ je definováno předpisem

(i) $\delta(p, 0, A) = \{(p, BA), (q, A)\}$,

(ii) $\delta(p, 0, B) = \{(p, BB)\}$,

(iii) $\delta(p, 1, B) = \{(p, e)\}$,

(iv) $\delta(q, 0, A) = \{(q, A), (q, e)\}$,

(v) $\delta(q, e, A) = \{(p, BB)\}$,

zkonstruujme algoritmem z důkazu předchozí věty zásobníkový automat $\mathscr{M}'$ s jediným stavem takový, že $N(\mathscr{M}) = N(\mathscr{M}')$. Položme $\mathscr{M}' = (\{p\}, \{0, 1\}, \Gamma', \delta', p, R, \emptyset)$ a sestrojme příslušné Γ' a δ'. Místo zásobníkových symbolů tvaru $\langle p, D, q\rangle$ budeme pro větší přehlednost psát ${}_pD_q$. Podle důkazu věty 4.5.3 tedy bude

$$\Gamma' = \{R, {}_pA_p, {}_pA_q, {}_qA_p, {}_qA_q, {}_pB_p, {}_qB_p, {}_pB_q, {}_qB_q\}$$

a instrukce

0. $\delta'(p, e, R) = \{(p, {}_pA_p), (p, {}_pA_q)\}$,

1. $\begin{cases} \delta'(p, 0, {}_pA_p) = \{(p, {}_pB_p\,{}_pA_p), (p, {}_pB_q\,{}_qA_p), (p, {}_qA_p)\}, \\ \delta'(p, 0, {}_pA_q) = \{(p, {}_pB_p\,{}_pA_q), (p, {}_pB_q\,{}_qA_q), (p, {}_qA_q)\}, \end{cases}$

$$\text{II.}\quad \begin{cases} \delta'(p, 0, {}_pB_p) = \{(p, {}_pB_p\ {}_pB_p), (p, {}_pB_q\ {}_qB_p)\}, \\ \delta'(p, 0, {}_pB_q) = \{(p, {}_pB_p\ {}_pB_q), (p, {}_pB_q\ {}_qB_q)\}, \end{cases}$$

$$\text{III.}\quad \delta'(p, 1, {}_pB_p) = \{(p, e)\},$$

$$\text{IV.}\quad \begin{cases} \delta'(p, 0, {}_qA_p) = \{(p, {}_qA_p)\}, \\ \delta'(p, 0, {}_qA_q) = \{(p, {}_qA_q), (p, e)\}, \end{cases}$$

$$\text{V.}\quad \begin{cases} \delta'(p, e, {}_qA_p) = \{(p, {}_pB_p\ {}_pB_p), (p, {}_pB_q\ {}_qB_p)\}, \\ \delta'(p, e, {}_qA_q) = \{(p, {}_pB_p\ {}_pB_q), (p, {}_pB_q\ {}_qB_q)\}. \end{cases}$$

Instrukce 0 vznikají podle pravidla 1 (z důkazu předchozí věty), instrukce I vznikají z instrukce (i), instrukce II z instrukce (ii) a instrukce V vznikají z instrukce (v), vše podle pravidla 3 z důkazu, instrukce III z instrukce (iii) podle pravidla 2 a instrukce IV z instrukcí (iv) podle pravidel 3 a 2.

Má-li automat jediný stav, potom udávání jeho stavu během výpočtu nepřináší žádnou informaci. Instrukce takového automatu určují pouze, jaký vstupní symbol byl přečten a jakým způsobem byl přepsán vrchní symbol zásobníku. Takovou informaci lze snadno zachytit bezkontextovým pravidlem a na tomto pozorování je založena i následující věta.

Věta 4.5.5. *Ke každému zásobníkovému automatu $\mathscr{M}$ lze sestrojit bezkontextovou gramatiku $\mathscr{G}$ tak, že $L(\mathscr{G}) = N(\mathscr{M})$.*

Důkaz. Bez újmy na obecnosti můžeme předpokládat, že $\mathscr{M}$ má jediný vnitřní stav. Pokud má totiž více stavů, lze jej podle předchozí věty převést na ekvivalentní automat s jediným stavem a tento automat můžeme zvolit jako východisko konstrukce. Budiž tedy

$$\mathscr{M} = (\{p\}, \Sigma, \Gamma, \delta, p, A, \emptyset).$$

Případným přejmenováním zásobníkových symbolů lze docílit, aby $\Gamma \cap \Sigma = \emptyset$. Sestrojme gramatiku

$$\mathscr{G} = (\Gamma, \Sigma, A, P),$$

kde P je definováno tímto předpisem:

$$B \to a\omega$$

patří do P, právě když

$$(p, \omega) \in \delta(p, a, B),$$

kde

$$a \in \Sigma \cup \{e\},\ B \in \Gamma \text{ a } \omega \in \Gamma^*.$$

Levé derivace podle gramatiky $\mathscr{G}$ přesně odpovídají výpočtům automatu $\mathscr{M}$. Přesněji: pro libovolné $u \in \Sigma^*$, $\eta \in \Gamma^*$ a $k \geqq 1$ je

(*) $A \Rightarrow_{\mathscr{G}}^{*} u\eta$ levou derivací délky k,

právě když $(p, u, A) \vdash_{\mathscr{M}}^{*} (p, e, \eta)$ výpočtem délky k.

O pravdivosti (*) se můžeme přesvědčit indukcí. Pro $k = 1$ plyne (*) bezprostředně z definice $\mathscr{G}$. Předpokládejme tedy platnost (*) pro jisté $l \geqq 1$. Z tohoto předpokladu dokážeme pravdivost (*) pro $k = l + 1$ takto:

$$A \Rightarrow^* uB\xi \Rightarrow v\eta$$

je levá derivace podle gramatiky $\mathscr{G}$ délky $l + 1$, právě když

$$A \Rightarrow^* uB\xi$$

je levá derivace délky l a v P existuje pravidlo

$$B \to av \quad (a \in \Sigma \cup \{e\},\ v \in \Gamma^*)$$

takové, že $ua = v$ a $v\xi = \eta$. To podle indukčního předpokladu a definice $\mathscr{G}$ nastane právě tehdy, když

$$(p, ua, A) \vdash_{\mathscr{M}}^{*} (p, a, B\xi)$$

výpočtem délky l a

$$(p, a, B\xi) \vdash_{\mathscr{M}} (p, e, v\xi),$$

tzn.

$$(p, ua, A) \vdash^* (p, e, \eta)$$

výpočtem délky $l + 1$.

Z (*) dostáváme jako speciální případ pro $\eta = e$:

$$u \in L(\mathscr{G}) \Leftrightarrow u \in N(\mathscr{M}).$$

Příklad 4.5.6. K zásobníkovému automatu $\mathscr{M}$ z př. 4.5.4 sestrojme gramatiku $\mathscr{G}$ takovou, že $L(\mathscr{G}) = N(\mathscr{M})$.

Použijeme automatu $\mathscr{M}'$ s jedním stavem zkonstruovaného v 4.5.4 a z něho okamžitě dostáváme

$$
\begin{aligned}
R &\to {}_pA_p \mid {}_pA_q,\\
{}_pA_p &\to 0\,{}_pB_p\,{}_pA_p \mid 0\,{}_pB_q\,{}_qA_p \mid 0\,{}_qA_p,\\
{}_pA_q &\to 0\,{}_pB_p\,{}_pA_q \mid 0\,{}_pB_q\,{}_qA_q \mid 0\,{}_qA_q,\\
{}_pB_p &\to 0\,{}_pB_p\,{}_pB_p \mid 0\,{}_pB_q\,{}_qB_p,\\
{}_pB_q &\to 0\,{}_pB_p\,{}_pB_q \mid 0\,{}_pB_q\,{}_qB_q,\\
{}_pB_p &\to 1,\\
{}_qA_p &\to 0\,{}_qA_p,\\
{}_qA_q &\to 0\,{}_qA_q \mid 0,\\
{}_qA_p &\to {}_pB_p\,{}_pB_p \mid {}_pB_q\,{}_qB_p,\\
{}_qA_q &\to {}_pB_p\,{}_pB_q \mid {}_pB_q\,{}_qB_q.
\end{aligned}
$$

Tuto gramatiku můžeme podle 4.1.5 zredukovat a dostáváme

$$
\begin{aligned}
R &\to {}_pA_p \mid {}_pA_q,\\
{}_pA_p &\to 0\,{}_pB_p\,{}_pA_p \mid 0\,{}_qA_p,\\
{}_pA_q &\to 0\,{}_pB_p\,{}_pA_q \mid 0\,{}_qA_q,\\
{}_pB_p &\to 1 \mid 0\,{}_pB_p\,{}_pB_p,\\
{}_qA_p &\to 0\,{}_qA_p \mid {}_pB_p\,{}_pB_p,\\
{}_qA_q &\to 0 \mid 0\,{}_qA_q \mid {}_pB_p\,{}_pB_p.
\end{aligned}
$$

Přejmenováním neterminálů $R = A$, ${}_pA_p = B$, ${}_pA_q = C$, ${}_pB_p = D$, ${}_qA_p = E$, ${}_qA_q = F$ vznikne přehlednější gramatika

$$
\begin{aligned}
A &\to B \mid C,\\
B &\to 0DB \mid 0E,\\
C &\to 0DC \mid 0F,\\
D &\to 1 \mid 0DD,\\
E &\to 0E \mid DD,\\
F &\to 0 \mid 0F \mid DD.
\end{aligned}
$$

Důsledek 4.5.7. *Pro libovolný jazyk L jsou tato tvrzení ekvivalentní:*

a) *L je bezkontextový jazyk.*

b) *$L = N(\mathcal{M})$ pro nějaký zásobníkový automat $\mathcal{M}$.*

c) *$L = N(\mathcal{M})$ pro nějaký zásobníkový automat $\mathcal{M}$ s jediným stavem.*

d) *$L = L(\mathcal{M})$ pro nějaký zásobníkový automat $\mathcal{M}$.*

Poznámka 4.5.8. Jelikož přechod od zásobníkového automatu s jediným stavem k bezkontextové gramatice je přímočarý, je snadné vytvořit na základě daného zásobníkového automatu $\mathcal{M}$ přímo ekvivalentní gramatiku, aniž by mezitím bylo třeba explicitně konstruovat automat $\mathcal{M}'$ s jediným stavem ekvivalentní automatu $\mathcal{M}$.

Jedním z bezprostředních důsledků vztahu mezi bezkontextovými jazyky a zásobníkovými automaty je uzavřenost třídy bezkontextových jazyků vůči průniku s regulárními jazyky.

Definice 4.5.9. Říkáme, že *třída jazyků $\mathscr{L}$ je uzavřena vůči průniku s regulárními jazyky*, jestliže pro libovolný jazyk $L \in \mathscr{L}$ a regulární jazyk R je $L \cap R \in \mathscr{L}$.

Věta 4.5.10. *Třída bezkontextových jazyků je uzavřena vůči průniku s regulárními jazyky. Třída jazyků rozpoznatelných deterministickými zásobníkovými automaty je také uzavřena vůči průniku s regulárními jazyky.*

Platnost této věty můžeme nahlédnout pomocí následující představy. Mějme zásobníkový automat $\mathcal{M}$ rozpoznávající jazyk L koncovým stavem a konečný automat $\mathcal{A}$ rozpoznávající jazyk R. Představíme-li si $\mathcal{M}$ jako zařízení s jednosměrnou vstupní páskou P_1, konečnou řídicí jednotkou C_1 a zásobníkem Z a automat $\mathcal{A}$ jako zařízení s jednosměrnou vstupní páskou P_2 a konečnou řídicí jednotkou C_2, lze z těchto dvou zařízení vytvořit jediné zařízení ztotožněním P_1 a P_2. Vznikne tak zařízení, které má jednu vstupní jednosměrnou pásku, jeden zásobník a soustavu dvou konečných řídicích jednotek C_1, C_2. Tato soustava ovšem nabývá pouze konečně mnoha stavů a lze na ni pohlížet jako na jednou jednotku. Vzniklé zařízení je tedy zásobníkový auto-

mat. Jestliže za počáteční stav tohoto automatu považujeme stav, kdy obě složky řídicí jednotky jsou ve svých počátečních stavech, a za koncové ty stavy, ve kterých jsou obě složky v koncových stavech, je vidět, že nově vzniklý automat rozpoznává $L \cap R$. Jelikož se jedná o zásobníkový automat, je $L \cap R$ bezkontextový.

Následující důkaz je pouze formálním vyjádřením této úvahy. Všimněte si v něm způsobu, jakým je vyřešen problém, že $\mathscr{M}$ může dělat e-kroky.

Důkaz. Budiž L bezkontextový jazyk rozpoznávaný pomocí koncového stavu zásobníkovým automatem

$$\mathscr{M} = (Q_1, \Sigma, \Gamma, \delta_1, q_1, Z_0, F_1)$$

a R budiž regulární jazyk rozpoznávaný konečným automatem

$$\mathscr{A} = (Q_2, \Sigma, \delta_2, q_2, F_2).$$

Zřejmě můžeme bez újmy na obecnosti předpokládat, že oba automaty mají stejnou vstupní abecedu. Sestrojíme zásobníkový automat $\mathscr{M}'$, který rozpoznává koncovým stavem jazyk $L \cap R$. Automat $\mathscr{M}'$ je definován takto:

$$\mathscr{M}' = (Q_1 \times Q_2, \Sigma, \Gamma, \delta, (q_1, q_2), Z_0, F_1 \times F_2),$$

kde δ je zadáno těmito vztahy:

$$((p', q'), \gamma) \in \delta((p, q), a, Z) \Leftrightarrow$$

$$\Leftrightarrow \text{buď}$$

$$a \in \Sigma \,\&\, (p', \gamma) \in \delta_1(p, aZ) \,\&\, \delta_2(q, a) = q',$$

nebo

$$a = e \,\&\, (p', \gamma) \in \delta_1(p, e, Z) \,\&\, q = q',$$

kde

$$p, p' \in Q_1,\ q, q' \in Q_2,\ a \in \Sigma \cup \{e\},\ \gamma \in \Gamma^*,\ Z \in \Gamma.$$

Čtenář (na základě předchozí úvahy) sám snadno ověří, že

$$L(\mathscr{M}') = L(\mathscr{M}) \cap L(\mathscr{A}).$$

Mimoto je z definice automatu $\mathscr{M}'$ okamžitě zřejmé, že $\mathscr{M}'$ je deterministický, je-li $\mathscr{M}$ deterministický.

4.6. Principy analýzy zdola

Syntaktická analýza metodou shora, s níž jsme se na intuitivní úrovni seznámili v čl. 4.3, je spjata s druhým základním postupem syntaktické analýzy, analýzou zdola. Tuto metodu a její vztah k dříve popsané metodě si budeme ilustrovat opět na příkladu gramatiky

(1) $A \to A + B$,

(2) $A \to B$,

(3) $B \to B * C$,

(4) $B \to C$,

(5) $C \to (A)$,

(6) $C \to a$

a analýze řetězu $a * (a + a)$. Metoda analýzy shora probírá vstupní řetěz zleva doprava a přitom vydává posloupnost čísel pravidel určujících levou derivaci vstupního řetězu.

Je zřejmé, že při prohlížení vstupního řetězu zprava doleva bychom obdobným způsobem získali pravou derivaci vstupního řetězu. Tento postup pro řetěz $a * (a + a)$ je zachycen v tab. 6. Tentokrát je výhodnější zapisovat obsah zásobníku tak, že pravý konec slova odpovídá vrcholu zásobníku.

Uveďme několik charakteristik popsané analýzy řetězu

$$a * (a + a):$$

1. vstupní řetěz je procházen zprava doleva;

2. postupně jsou vydávány členy posloupnosti 2, 3, 5, 1, 4, 6, 2, 4, 6, 4, 6, tj. posloupnosti určující pravou derivaci analyzovaného řetězu;

3. odpovídající derivační strom (viz obr. 38) je procházen shora dolů (levým dolním rohem konče).

Představme si, že celý proces provedeme pozpátku. Přitom zřejmě bude

1. vstupní řetěz procházen zleva doprava;

2. postupně budou vydávány členy posloupnosti 6, 4, 6, 4, 2, 6, 4, 1, 5, 3, 2 určující pozpátku pravou derivaci;

Čtený symbol	Obsah zásobníku	Prováděná operace	Číslo pravidla
–	A	přepsání (vrcholu zásobníku) podle pravidla 2	2
–	B	přepsání podle pravidla 3	3
–	$B * C$	přepsání podle pravidla 5	5
)	$B * (A)$	terminální symbol na vrcholu zásobníku se srovná se vstupním symbolem (krácení)	–
–	$B * (A$	přepsání podle pravidla 1	1
–	$B * (A + B$	přepsání podle pravidla 4	4
–	$B * (A + C$	přepsání podle pravidla 6	6
a	$B * (A + a$	krácení	–
+	$B * (A +$	krácení	–
–	$B * (A$	přepsání podle pravidla 2	2
–	$B * (B$	přepsání podle pravidla 4	4
–	$B * (C$	přepsání podle pravidla 6	6
a	$B * (a$	krácení	–
(	$B * ($	krácení	–
*	$B *$	krácení	–
–	B	přepsání podle pravidla 4	4
–	C	přepsání podle pravidla 6	6
a	a	krácení	–
slovo přečteno	–		

Tab. 6

3. odpovídající derivační strom bude procházen zdola nahoru (levým dolním rohem počínaje).

Základní operace analýzy shora (přepsání vrcholu zásobníku a krácení) se při zpětném průběhu projeví jako dva nové typy operací:

a) přesun nového vstupního symbolu do zásobníku (odpovídá krácení);

b) redukce několika vrchních symbolů zásobníku na jediný symbol (odpovídá přepsání vrcholu zásobníku).

Podrobněji: jestliže v původním procesu byl vrchní symbol zásobníku nahrazen řetězem ω podle pravidla $D \to \omega$, bude opačná akce spočívat v tom, že několik vrchních symbolů zá-

zásobníku tvořících řetěz ω bude podle uvedeného pravidla redukováno na D.

Analýza začíná s prázdným zásobníkem a při úspěšném průběhu (tj. přijetí slova) je po přečtení celého vstupního řetězu v zásobníku pouze počáteční symbol.

Podle způsobu procházení derivačním stromem je tento způsob analýzy nazýván *analýzou zdola*. Místo procházení stromu je možné si představovat jeho postupné vytváření.

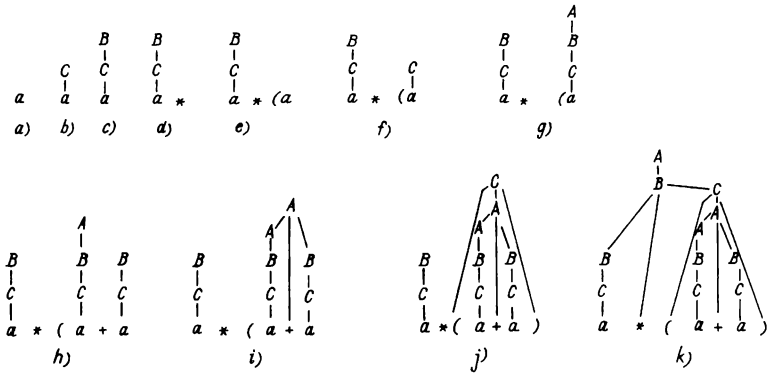
Čtený symbol	Obsah zásobníku	Prováděná operace	Číslo pravidla	Viz obr. 40
a		načtení do zásobníku	–	–
	a	redukce podle pravidla 6	6	a)
	C	redukce podle 4	4	b)
$*$	B	přesun do zásobníku	–	c)
$($	$B*$	přesun do zásobníku	–	d)
a	$B*($	přesun do zásobníku	–	–
	$B*(a$	redukce podle 6	6	e)
	$B*(C$	redukce podle 4	4	f)
	$B*(B$	redukce podle 2	2	–
$+$	$B*(A$	přesun do zásobníku	–	g)
a	$B*(A+$	přesun do zásobníku	–	–
	$B*(A+a$	redukce podle 6	6	–
	$B*(A+C$	redukce podle 4	4	–
	$B*(A+B$	redukce podle 1	1	h)
$)$	$B*(A$	přesun do zásobníku	–	i)
	$B*(A)$	redukce podle 5	5	–
	$B*C$	redukce podle 3	3	j)
	B	redukce podle 2	2	–
	A	konec analýzy	–	k)

Tab. 7

Průběh popsaného typu analýzy řetězu $a*(a+a)$ podle výchozí gramatiky je zachycen v tab. 7 a několik stadií konstrukce derivačního stromu je na obr. 40.

Zcela analogicky jako v čl. 4.3 jsme zatím odsunuli stranou otázku, jakým způsobem se určí správné pokračování v případě, že existuje několik možných. Jedno z řešení tohoto problému

je vyloženo v kap. 5 v části o LR gramatikách. Na této úrovni se opět spokojíme s nedeterministickým postupem. Snadno ovšem můžeme určit, jakou třídu jazyků lze tímto postupem analyzovat.



Obr. 40

Stačí k tomu dvě jednoduché úvahy.

1. Pro modifikované zásobníkové automaty, které procházejí vstupní slova zprava, lze analogicky jako v čl. 4.5 ukázat (s použitím pravých na místě levých derivací), že rozpoznávají právě bezkontextové jazyky.

2. Jestliže od zásobníkového automatu $\mathscr{M}_1$ analyzujícího vstupní slovo zprava přejdeme k automatu $\mathscr{M}_2$ realizujícímu výpočty $\mathscr{M}_2$ pozpátku (viz přechod od tab. 6 k tab. 7), potom úspěšné (přijímající) výpočty $\mathscr{M}_2$ jsou právě všechny pozpátku provedené přijímající výpočty automatu $\mathscr{M}_1$. Automat $\mathscr{M}_2$ proto přijímá stejný jazyk jako $\mathscr{M}_1$.

Shrnutím obou pozorování dostáváme, že nedeterministickou analýzou zdola lze analyzovat právě bezkontextové jazyky.

4.7. Chomského normální forma, lemma o vkládání

V tomto článku ukážeme, že všechny bezkontextové jazyky je možné generovat gramatikami v jistém speciálním tvaru. Derivační stromy příslušné k těmto gramatikám budou mít uniformní

tvar usnadňující různé kvantitativní odhady, jak nastíníme v důkazu tzv. lemmatu o vkládání.

Definice 4.7.1. *Bezkontextová gramatika je v Chomského normální formě*, právě když všechna její pravidla jsou tvaru $X \to YZ$ nebo $X \to a$, kde X, Y, Z jsou neterminály a a je terminální symbol.

Z definice je patrné, že jazyk generovaný gramatikou v Chomského normální formě nemůže obsahovat prázdné slovo. Ukážeme, že až na tuto výhradu lze každý bezkontextový jazyk generovat nějakou gramatikou v Chomského normální formě.

Věta 4.7.2. *Ke každému bezkontextovému jazyku L existuje gramatika $\mathscr{G}$ v Chomského normální formě taková, že*

$$L - \{e\} = L(\mathscr{G}).$$

Důkaz. Konstrukce popisované v důkazu věty budeme průběžně ilustrovat na jednoduché gramatice.

Budiž $\mathscr{G}' = (\Pi', \Sigma, A, P')$ gramatika generující $L - \{e\}$. Díky větě 3.2.8 můžeme za $\mathscr{G}'$ zvolit nevypouštějící gramatiku. Pro náš příklad to bude tato gramatika:

$$A \to B \mid C,$$
$$B \to 0B1 \mid 01,$$
$$C \to D \mid E,$$
$$D \to 1D0 \mid 1,$$
$$E \to 0E \mid 0.$$

Nejprve z gramatiky odstraníme všechna pravidla typu $X \to Y$, kde X, Y jsou neterminály. Při tom postupujeme takto: Ke každému neterminálu Z najdeme všechna X taková, že

$$X \Rightarrow X_1 \Rightarrow X_2 \Rightarrow \ldots \Rightarrow X_n = Z$$

pro nějaké neterminály

$$X_1, \ldots, X_n, \quad n \geqq 1.$$

S každým pravidlem $Z \to \eta$ pak do množiny pravidel zahrneme také $X \to \eta$. Když takto rozšíříme množinu pravidel, odstraníme

všechna pravidla typu $X \to Y$. V našem příkladu tak dostáváme tuto gramatiku:

$$A \to 0B1 \mid 01,$$
$$B \to 0B1 \mid 01,$$
$$A \to 1D0 \mid 1,$$
$$C \to 1D0 \mid 1,$$
$$D \to 1D0 \mid 1,$$
$$A \to 0E \mid 0,$$
$$C \to 0E \mid 0,$$
$$E \to 0E \mid 0.$$

Je zřejmé, že uvedenou konstrukcí vznikne gramatika ekvivalentní s původní gramatikou. Použití nově přidaných pravidel se totiž ve výchozí gramatice dají simulovat několika kroky derivace. Naopak úseky derivace podle první gramatiky užívající pouze pravidel typu $X \to Y$ jsou následovány aplikací pravidla typu $Z \to \eta$, kde $|\eta| \geqq 2$ nebo $\eta \in \Sigma$. Celý takový úsek lze nahradit jedním z nově přidaných pravidel.

Než postoupíme dále, všimněme si, že v našem příkladu vznikla po popsané transformaci gramatika, která není redukovaná (nedosažitelný neterminál C). Proto ji pro zjednodušení zredukujme takto:

$$A \to 0B1 \mid 01 \mid 1D0 \mid 1 \mid 0E \mid 0,$$
$$B \to 0B1 \mid 01,$$
$$D \to 1D0 \mid 1,$$
$$E \to 0E \mid 0.$$

Po popsané úpravě dostáváme gramatiku, u níž všechna pravidla s pravou stranou kratší než 2 jsou ve tvaru vyhovujícím Chomského normální formě. Jsou totiž tvaru $X \to a$, kde X je neterminál a a terminál.

V dalším výkladu se tedy stačí zabývat pouze pravidly typu $X \to \eta$, kde $|\eta| \geqq 2$. Tato pravidla upravíme tak, aby se na jejich pravé straně vyskytovaly pouze neterminály. Toho dosáhneme

přidáním nového neterminálního symbolu pro každý terminál. Neterminál Y takto přidaný k terminálu y budeme nazývat zástupným neterminálem za y. Do množiny pravidel přidáme pravidla $Y \to y$ a v každém pravidle s pravou stranou délky alespoň 2 nahradíme všechny výskyty terminálů jejich zástupnými neterminály.

V našem příkladu přidejme zástupný neterminál F za 0 a G za 1. Dostaneme tak

$$A \to FBG \mid FG \mid GDF \mid 1 \mid FE \mid 0,$$
$$B \to FBG \mid FG,$$
$$D \to GDF \mid 0,$$
$$E \to FE \mid 0,$$
$$F \to 0,$$
$$G \to 1.$$

Uvedenou transformací opět dostaneme gramatiku ekvivalentní s gramatikou původní. Všechna pravidla získané gramatiky, která mají pravou stranu délky nejvýše 2, už vyhovují tvaru Chomského normální formy. Zbývá tedy upravit pravidla typu

$$X \to X_1 X_2 \ldots X_n, \quad n \geqq 3.$$

Každé takovéto pravidlo nahradíme soustavou pravidel

$$X \to X_1 Y_1,$$
$$Y_1 \to X_2 Y_2,$$
$$\ldots\ldots\ldots$$
$$Y_{n-2} \to X_{n-1} X_n,$$

kde $Y_1, \ldots, Y_{n-2}$ jsou další nově přidané neterminály nevyskytující se v jiných pravidlech. Díky tomu se v derivaci uplatní vždy celá soustava, a to přesně se stejným efektem, jako by mělo přepsání podle vyřazeného pravidla. Vzniklá gramatika je proto opět ekvivalentní s gramatikou původní.

V našem příkladu tak dostaneme tuto gramatiku:

$$A \to FA_1 \mid FG \mid GA_2 \mid 1 \mid FE \mid 0,$$
$$A_1 \to BG,$$
$$A_2 \to DF,$$
$$B \to FB_1 \mid FG,$$
$$B_1 \to BG,$$
$$D \to GD_1 \mid 0,$$
$$D_1 \to DF,$$
$$E \to FE \mid 0,$$
$$F \to 0,$$
$$G \to 1.$$

Poznámka 4.7.3. Každý derivační strom podle gramatiky v Chomského normální formě má tyto dvě vlastnosti:

a) z každého vnitřního uzlu vycházejí dvě nebo jedna hrana;

b) z uzlu vychází jediná hrana právě tehdy, jestliže tato hrana vchází do listu.

Uvedené vlastnosti oceníme při důkazu následující věty, která pro bezkontextové jazyky hraje podobnou roli jako věta 1.3.3 pro regulární jazyky. Poskytuje jednoduchý a elegantní nástroj pro situace, kdy je třeba ověřit, že jistý jazyk není bezkontextový.

Věta 4.7.4 (Lemma o vkládání).* *Nechť L je bezkontextový jazyk. Potom existují přirozená čísla m, n taková, že každé slovo $z \in L$ délky větší než n může být napsáno ve tvaru*

$$z = uvwxy$$

tak, že

1. $uv^iwx^iy \in L$ *pro všechna* $i \geqq 0$,
2. *alespoň jedno ze slov* v, x *je neprázdné*,
3. $|vwx| \leqq m$.

---

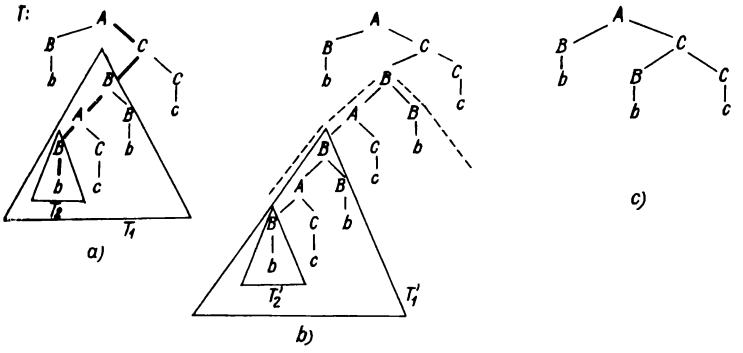
*) V anglické literatuře je tato věta známa jako „pumping lemma" pro bezkontextové jazyky.

Důkaz věty v plné obecnosti je možné najít v knize [20]. Zde probereme konkrétní příklad a na něm osvětlíme všechny podstatné úvahy, o které se důkaz opírá.

Uvažujme bezkontextový jazyk L generovaný gramatikou

$$A \to BC \mid a,$$
$$B \to BA \mid AB \mid b,$$
$$C \to BC \mid c.$$

Obr. 41a) zachycuje derivační strom T pro slovo *bbcbc*. V tomto stromě se na některých cestách od kořene k listům opakovaně vyskytuje stejný neterminál. Příkladem jsou dva výskyty neterminálu B na cestě, která je na obr. 41a) vyznačena



Obr. 41

silně. První výskyt tohoto neterminálu je kořenem podstromu popisujícího strukturu řetězu *bcb* a označeného na obrázku jako T_1, druhý výskyt je kořenem podstromu T_2 popisujícího strukturu *b*. Z obrázku je možné vyčíst, že

(*) $A \Rightarrow^* bBc$ (T_1 je podstromem T),

(**) $B \Rightarrow^* Bcb$ (T_2 je podstromem T_1),

(***) $B \Rightarrow^* b$ (strom T_2).

Jestliže na obr. 41a) nahradíme podstrom T_2 podstromem T_1,

dostaneme strom z obr. 41b) popisující řetěz $bb(cb)^2\,c$. Jinými slovy, od derivace odpovídající obr. 41a):

$$A \Rightarrow^* bBc \Rightarrow^* bBcbc \Rightarrow^* bbcbc$$

přejdeme [opakovaným užitím (**)] k derivaci odpovídající obr. 41b):

$$A \Rightarrow^* bBc \Rightarrow^* bBcbc \Rightarrow^* bBcbcbc \Rightarrow^* bb(cb)^2\,c\,.$$

Výsledkem je zdvojení podřetězu cb. Kdybychom v obr. 41b) opět nahradili podstrom T_2' podstromem T_1', dostali bychom derivační strom řetězu $bb(cb)^3\,c$ a iterací tohoto postupu můžeme získat derivační strom pro $bb(cb)^i\,c$ pro každé $i \geqq 1$. Jestliže naopak ve stromu z obr. 41a) nahradíme podstrom T_1 podstromem T_2, dostaneme derivační strom z obr. 41c), který popisuje řetěz bbc, tj. $bb(cb)^0\,c$. Je tedy $bb(cb)^i\,c \in L$ pro všechna $i \geqq 0$.

Označíme-li $u = b$, $v = e$, $w = b$, $x = cb$, $y = c$, máme $bbcbc = uvwxy$ a $uv^iwx^iy \in L$ pro $i \geqq 0$, tzn. našli jsme rozklad slova $bbcbc$ splňující tvrzení 1 věty, ale i tvrzení 2, neboť zde $x \neq e$.

Přesvědčme se, že takový rozklad je možné najít pro každé dostatečně dlouhé slovo $z \in L$. Okamžitě je vidět, že popsaný postup lze použít na každý derivační strom, v němž jsou na některé cestě od kořene k vrcholu dva různé vrcholy ohodnoceny stejným neterminálem. To samozřejmě nastane na každé cestě délky větší, než je celkový počet k neterminálů gramatiky. Na cestě délky l totiž leží $l + 1$ vrcholů, z toho l ohodnocených neterminály. V našem případě je proto určitě možné postup aplikovat na libovolný derivační strom výšky větší než 3.

Derivačních stromů výšky omezené číslem k je pro každou gramatiku konečně mnoho, a proto existuje takové číslo n, že každý derivační strom pro libovolné slovo délky větší než n už nutně obsahuje cestu delší než k. (Snadno se spočítá, že pro náš případ je $n = 4$, obecně pro gramatiku v Chomského normální formě s k neterminály $n = 2^{k-1}$.)

Tím jsme ověřili, že každé slovo z jazyka L délky větší než n je možné psát ve tvaru $uvwxy$ tak, že platí tvrzení 1 věty. Tvrzení 2 je v tom případě také splněno z důvodu, který je vidět z obr. 41a).

Hrana vycházející z kořene podstromu T_1, která nesměřuje ke kořeni podstromu T_2 (v tomto případě pravá), vchází do vrcholu ohodnoceného neterminálem (zde B). Jelikož gramatika je v Chomského normální formě, je z tohoto neterminálu vygenerováno neprázdné slovo, které je podslovem slova v nebo (jako v našem případě) podslovem slova x.

Zbývá se přesvědčit, že rozklad slova z je možné zvolit tak, aby platilo i tvrzení 3, tzn. aby podslovo vwx mělo délku kratší než jistá předem daná konstanta závislá pouze na gramatice $\mathscr{G}$.

Zhruba řečeno, úsek vwx je tím kratší, čím je výška k němu příslušného podstromu menší. Tento podstrom je však možné zvolit tak, aby žádná jeho cesta neměla délku větší než $k + 1$. Proto lze dosáhnout toho, aby $|vwx| \leqq m$, kde $m = 2^k$ (pro gramatiku v Chomského normální formě).

Přesvědčili jsme se, že zvolená gramatika tvrzení splňuje. Naše úvahy však platí pro libovolnou gramatiku v Chomského normální formě, tedy podle 4.7.2 má požadované vlastnosti každý jazyk $L - \{e\}$, kde L je libovolný bezkontextový jazyk. Jelikož tvrzení věty hovoří pouze o slovech kladné délky, plyne z její platnosti pro $L - \{e\}$ i platnost pro L.

Lemma o vkládání nejčastěji slouží k důkazu tvrzení, že nějaký jazyk není bezkontextový.

Příklad 4.7.5. Ukažme, že jazyk $L = \{a^n b^n c^n; n \geqq 1\}$ není bezkontextový.

Předpokládejme opak, tj. že L je bezkontextový a že tedy existují přirozená čísla m, n s vlastnostmi uvedenými v lemmatu o vkládání. Zvolme $l > n/3$. Potom slovo $z_1 = a^l b^l c^l$ je možné psát ve tvaru $z_1 = uvwxy$, kde $v \neq e$ nebo $x \neq e$, a každé slovo $z_i = uv^i wx^i y$ leží v L. Protože ve všech slovech z L se symboly a, b, c vyskytují v abecedním pořádku a protože $z_i \in L$ i pro $i > 1$, může každé ze slov v, x obsahovat pouze stejné symboly (tj. např. v obsahuje pouze b, x obsahuje pouze c; kdyby totiž např. $v = aab$, potom by už z_2 obsahovalo podslovo $aabaab$ a nemohlo by patřit do L). Ve slovech z_i proto s rostoucím i roste počet nejvýše dvou druhů symbolů (obsažených ve v a x), zatímco symbolů třetího druhu je stále stejný počet. (Přitom délka

z_i s rostoucím i roste, protože v nebo x je neprázdné.) Tím jsme dospěli ke sporu s definicí L. Jazyk L tedy není bezkontextový.

Využili jsme nyní jen tvrzení 1 a 2 lemmatu o vkládání. Ukažme si o něco komplikovanější příklad, ve kterém se využije i tvrzení 3.

Příklad 4.7.6. Ukažme, že jazyk $L = \{ww;\ w \in \{0, 1\}^*\}$ není bezkontextový.

Opět provedeme důkaz sporem. Předpokládejme, že L je bezkontextový a že tedy k němu existují čísla m, n s vlastnostmi uvedenými v lemmatu o vkládání. Zvolme $k > \max(m, n)$ a označme $w_1 = w_3 = 0^k$, $w_2 = w_4 = 1^k$ a $z = w_1w_2w_3w_4$, tj. $z = 0^k1^k0^k1^k$. Zřejmě $z \in L$. Uvažujme jeho libovolný rozklad $z = uvwxy$ splňující podmínky věty 4.7.4. Jelikož $|vwx| \leqq m$ a z jsme zvolili tak, že $|w_i| > m$ pro každé i, $1 \leqq i \leqq 4$, nemůže podslovo vwx ve slově z zasahovat do více než dvou sousedních úseků w_i. Nechť zasahuje např. do úseků w_1, w_2. Slovo $uwy = uv^0wx^0y$ má podle věty 4.7.4 také patřit do L, ale přitom $uwy = 0^r1^s0^k1^k$, kde $0 < r \leqq k$, $0 < s \leqq k$ a $r + s < k + k$ (protože alespoň jedno z vypuštěných podslov v, x je neprázdné). Slovo tohoto tvaru ovšem nevyhovuje definici jazyka L. Podobně se vyloučí možnosti, že vwx zasahuje nejvýš do w_2w_3 nebo w_3w_4. Tím je předpoklad, že L je bezkontextový, doveden ke sporu.

ROZŠIŘUJÍCÍ ČÁST

4.8. Uzávěrové vlastnosti

V kapitole 2 jsme naznačili několik možností, jak využít uzávěrových vlastností třídy regulárních jazyků. Podobný užitek přináší znalost uzávěrových vlastností u bezkontextových jazyků. Základní přehled podává tab. 8. Značka + (resp. −) označuje, že příslušná třída jazyků je (resp. není) vůči uvedené operaci uzavřena. Ve vedlejším sloupci je pak vždy uveden odkaz na pramen, ve kterém je možné nalézt důkaz dotyčné vlastnosti. K tabulce jsou připojeny komentáře.

Uveďme definice několika pojmů, s kterými jsme se zatím nesetkali.

Operace	Bezkontextové jazyky	Pramen	Deterministické jazyky	Pramen
sjednocení	+	[20]	–	4.8.8
průnik	–	4.8.7	–	4.8.7
doplněk	–	4.8.8	+	[20]
zřetězení	+	[20]	–	4.8.11
iterace	+	[20]	–	4.8.11
zrcadlový obraz	+	[20]	–	4.8.10
průnik s regulárním jazykem	+	4.5.10	+	4.5.10
substituce	+	[20]	–	4.8.9
homomorfismus	+	[20]	–	4.8.9
inverzní homomorfismus	+	4.8.13	+	4.8.10
sekvenční zobrazení	+	4.8.16	–	4.8.9
inverzní sekvenční zobrazení	+	4.8.16	–	4.8.10
pravý kvocient s regulárním jazykem	+	[20]	+	[20]
levý kvocient s regulárním jazykem	+	4.8.10	–	4.8.10
Min	–	4.8.12	+	[20]
Max	–	4.8.12	+	[20]

Tab. 8

Definice 4.8.1. Řekneme, že *třída jazyků* $\mathscr{L}$ *je uzavřena vůči*

1. *substituci* (srov. 2.2.7), jestliže pro každé $L \in \mathscr{L}$ a pro každou substituci $\sigma: \Sigma^* \to \mathscr{P}(\Delta^*)$ takovou, že pro všechna $a \in \Sigma$ je $\sigma(a) \in \mathscr{L}$, platí $\sigma(L) \in \mathscr{L}$;

2. *homomorfismu* (resp. *inverznímu homomorfismu*), jestliže pro každý homomorfismus h a každé $L \in \mathscr{L}$ je $h(L) \in \mathscr{L}$ [resp. $h^{-1}(L) \in \mathscr{L}$];

3. *pravému* (resp. *levému*) *kvocientu s regulárními jazyky*, jestliže pro každý regulární jazyk R a každé $L \in \mathscr{L}$ je $L/R \in \mathscr{L}$ (resp. $R \backslash L \in \mathscr{L}$).

Podobně se definuje uzavřenost i vůči několika dalším operacím, které nyní zavedeme.

Definice 4.8.2. *Zobecněným sekvenčním strojem* rozumíme šestici

$$\mathscr{S} = (Q, \Sigma, \Delta, \delta, q_0, F),$$

kde Q, Σ, Δ, q_0 a F jsou po řadě stavový prostor, vstupní a výstupní abeceda (vše konečné množiny), počáteční stav a množina koncových stavů, δ je zobrazení z $Q \times \Sigma$ do množiny všech konečných podmnožin množiny $Q \times \Delta^*$. Zobrazení δ rozšíříme přirozeným způsobem na zobrazení

$$\delta: Q \times \Sigma^* \to \mathscr{P}(Q \times \Delta^*)$$

takto:

1. $\delta(q, e) = \{(q, e)\}$,

2. $\delta(q, wa) = \{(p, v);$

v lze psát ve tvaru $v = v_1 v_2$ tak, že existuje $\hat{p} \in Q$, pro které $(\hat{p}, v_1) \in \delta(q, w)$ a $(p, v_2) \in \delta(\hat{p}, a)\}$.

Definujme pro libovolné slovo w a jazyk L:

$$S(w) = \{u;\ \text{existuje } p \in F \text{ tak, že } (p, u) \in \delta(q_0, w)\},$$

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

Takto získané zobrazení budeme nazývat *sekvenčním zobrazením.*

Podobně definujeme inverzní zobrazení tímto předpisem:

$$S^{-1}(w) = \{u;\ w \in S(u)\},$$
$$S^{-1}(L) = \bigcup_{w \in L} S^{-1}(w).$$

Takové zobrazení se nazývá *inverzní sekvenční zobrazení.*

Jestliže v definici 4.8.2 navíc požadujeme, aby

$$|\delta(q, x)| \leqq 1$$

pro každé $q \in Q$ a $x \in \Sigma$, dostáváme *deterministický zobecněný sekvenční stroj, deterministické sekvenční zobrazení* a *inverzní deterministické sekvenční zobrazení.*

Příklad 4.8.3. Budiž

$$\mathscr{S} = (Q, \Sigma, \Delta, \delta, q_0, F)$$

zobecněný sekvenční stroj s

$$Q = \{q_0, q_1, q_2, q_3\},\ \Sigma = \{a, b\},$$
$$\Delta = \{a, b, c, d\},\ F = \{q_3\}$$

a δ definovanou takto:

$$\delta(q_0, a) = \{(q_0, a), (q_1, a)\},$$
$$\delta(q_1, a) = \{(q_1, b)\},$$
$$\delta(q_1, b) = \{(q_2, c)\},$$
$$\delta(q_2, b) = \{(q_2, c), (q_3, dd)\},$$
$$\delta(q_3, b) = \{(q_3, dd)\}.$$

Potom např. pro $L_1 = \{a^i b^i;\ i \geqq 1\}$ je

$$S(L_1) = \{a^i b^j c^k d^{2l};\ j \geqq 0,\ i, k, l \geqq 1,\ i + j = k + l\}$$

a pro $L_2 = \{a, b, c, d\}^*$ je

$$S^{-1}(L) = \{a^i b^j;\ i, j \geqq 1\}.$$

Definice 4.8.4. Pro libovolný jazyk L definujme jazyky

$$\mathrm{Min}\,(L) = \{w;\ w \in L\ \&\ [w = uv\ \&\ v \neq e \Rightarrow u \notin L]\},$$
$$\mathrm{Max}\,(L) = \{w;\ w \in L\ \&\ [u \neq e \Rightarrow wu \notin L]\}.$$

Jazyk $\mathrm{Min}(L)$ se tedy skládá ze slov jazyka L, která nemají vlastní prefix patřící do L. Podobně $\mathrm{Max}(L)$ je tvořen slovy z L, která nelze prodloužit na žádné slovo patřící do L.

Příklad 4.8.5. Nechť

$$L = \{a^i b^j;\ 1 \leqq i \leqq j \leqq 2i\}.$$

Potom

$$\mathrm{Min}(L) = \{a^n b^n;\ n \geqq 1\},$$
$$\mathrm{Max}(L) = \{a^n b^{2n};\ n \geqq 1\}.$$

Komentáře.

4.8.6. Uzavřenost třídy bezkontextových jazyků vůči sjednocení, zřetězení, iteraci, zrcadlovému obrazu, substituci a homomorfismu jsou velmi jednoduché výsledky a čtenář si je může sám dokázat jako cvičení.

Dokázat, že bezkontextové jazyky jsou uzavřeny vůči pravému kvocientu s regulárním jazykem a že deterministické jazyky jsou uzavřeny vůči operaci Min je o něco obtížnější cvičení, zatímco výsledky o uzavřenosti deterministických jazyků vůči doplňku a uzavřenosti vůči operaci Max se opírají o značně komplikovaný důkaz.

4.8.7. Skutečnost, že bezkontextové ani deterministické jazyky nejsou uzavřeny vůči průniku, plyne z pozorování, že průnikem dvou deterministických jazyků

$$L_1 = \{a^i b^j c^k;\ i = j\},$$
$$L_2 = \{a^i b^j c^k;\ j = k\}$$

(reprezentujte je sami deterministickými zásobníkovými automaty) vznikne jazyk

$$L_1 \cap L_2 = \{a^n b^n c^n;\ n \geqq 1\},$$

o němž podle 4.7.5 víme, že není bezkontextový.

4.8.8. Bezkontextové jazyky nejsou uzavřeny vůči doplňku, jak plyne z de Morganova pravidla

$$L_1 \cap L_2 = -(-L_1 \cup -L_2)$$

a skutečnosti, že jsou uzavřeny vůči sjednocení a nejsou uzavřeny vůči průniku.

Z obdobného důvodu (uzavřenost vůči doplňku, neuzavřenost vůči průniku, de Morganovo pravidlo) nejsou deterministické jazyky uzavřeny vůči sjednocení.

4.8.9. Díky uzavřenosti deterministických jazyků vůči operaci Min se dá poměrně snadno dokázat, že bezkontextový jazyk

$$L_1 = \{ww^R;\ w \in \{a, b\}^*\}$$

není deterministický (viz např. skripta [22]). Jelikož jazyk

$$L_2 = \{wcw^R;\ w \in \{a, b\}^*\}$$

zjevně deterministický je, a protože homomorfismus h daný předpisem $h(a) = a$, $h(b) = b$, $h(c) = e$ převádí L_2 na L_1, není třída deterministických jazyků uzavřena vůči homomorfismu, a tím spíše ani vůči substituci či sekvenčnímu zobrazení.

4.8.10. Jiným příkladem bezkontextového jazyka, který není deterministický, je jazyk

$$L_1 = \{a^i b^j c^k;\ i = j \vee j = k\}$$

(viz skripta [22]). Naproti tomu jazyk

$$L_2 = \{fa^i b^j c^k;\ i = j\} \cup \{ga^i b^j c^k;\ j = k\}$$

zřejmě deterministický je. Protože

$$L_1 = \{f, g\} \backslash L_2 ,$$

není třída deterministických jazyků uzavřena vůči levému kvocientu s regulárním jazykem. Jelikož levý kvocient je možné vyjádřit pomocí operace pravého kvocientu a zrcadlového obrazu (srov. 2.2.18) a jelikož deterministické jazyky jsou uzavřeny vůči pravému kvocientu, nejsou deterministické jazyky uzavřeny vůči zrcadlovému obrazu. Na druhé straně, kombinace zrcadlového obrazu a pravého kvocientu s regulárním jazykem dává uzavřenost bezkontextových jazyků vůči levému kvocientu s regulárním jazykem.

Nedeterministický jazyk $f \,.\, L_1$ je možné získat z L_2 inverzním

nedeterministickým sekvenčním zobrazením (které zobrazuje f na f nebo g, a na a, b na b). Proto také nejsou deterministické jazyky uzavřeny vůči tomuto typu zobrazení. Jsou ovšem (viz kniha [20]) uzavřeny vůči inverznímu deterministickému sekvenčnímu zobrazení. Tím spíše pak vůči inverznímu homomorfismu.

4.8.11. Použijeme deterministického jazyka

$$L_1 = \{a^i b^j c^k;\ i = j\} \cup \{g a^i b^j c^k;\ j = k\}$$

(srov. s L_2 z 4.8.10) a deterministického jazyka $L_2 = \{g\}^*$. Zřetězením $L_2 . L_1$ vzniká jazyk, který není deterministický. Dále je zřejmě $L_1 . L_2$ deterministický, ale $(L_1 . L_2)^*$ deterministický není.

4.8.12. Neuzavřenost třídy bezkontextových jazyků vůči operacím Min a Max je patrná z příkladů bezkontextových jazyků

$$L_1 = \{a^i b^j c^k;\ k \geqq \min(i, j)\},$$
$$L_2 = \{a^i b^j c^k;\ k \leqq \max(i, j)\}.$$

Platí, že

$$\mathrm{Min}(L_1) = \{a^i b^j c^k;\ k = \min(i, j)\},$$
$$\mathrm{Max}(L_2) = \{a^i b^j c^k;\ k = \max(i, j)\},$$

a ani jeden z těchto jazyků není bezkontextový, jak se snadno dokáže z lemmatu o vkládání 4.7.4.

4.8.13. Zdaleka ne všechny operace uvedené v tab. 8 jsou na sobě nezávislé. Jestliže je např. dokázána uzavřenost třídy jazyků vůči průniku a doplňku, plyne z toho uzavřenost vůči sjednocení. Podobně z uzavřenosti třídy bezkontextových jazyků vůči substituci, homomorfismu a průniku s regulárním jazykem je možné dokázat její uzavřenost vůči inverznímu homomorfismu.

Pro libovolný homomorfismus h: $\Sigma^* \to \Delta^*$ totiž můžeme definovat abecedu $\bar{\Sigma}$ dvojníků k symbolům z Σ a dále

1. substituci σ předpisem

$$\sigma(c) = \bar{\Sigma}^* c \bar{\Sigma}^*$$

pro každé $c \in \Delta$,

2. regulární jazyk

$$L_1 = \{\bar{a}w;\ \bar{a} \in \bar{\Sigma},\ w \in \Delta^* \text{ a } h(a) = w\}$$

(také L_1^* je samozřejmě regulární jazyk),

3. homomorfismus $h_1 : (\bar{\Sigma} \cup \Delta)^* \to \Sigma^*$ předpisem

$$h_1(\bar{a}) = a \quad \text{pro všechna } \bar{a} \in \bar{\Sigma},$$

$$h_1(c) = e \quad \text{pro všechna } c \in \Delta.$$

Potom z uzavřenosti třídy $\mathscr{L}$ vůči zmíněným operacím plyne, že pro každý jazyk $L \in \mathscr{L}$ je

$$h_1(\sigma(L) \cap L_1^*) \in \mathscr{L}.$$

Přitom ale zřejmě platí, že

$$h_1(\sigma(L) \cap L_1^*) = h^{-1}(L).$$

4.8.14. Prohloubením podobných úvah vznikla celá rozsáhlá partie algebraické teorie formálních jazyků, zabývající se např. zkoumáním tzv. abstraktních tříd jazyků. Pojem abstraktní třídy jazyků vznikl vytypováním několika uzávěrových vlastností, které mají všechny čtyři třídy jazyků v Chomského hierarchii a které jsou splněny i četnými dalšími zajímavými třídami jazyků. V definici jsou vybrány co možná nejjednodušší uzávěrové vlastnosti, které implikují vlastnosti další.

Definice 4.8.15. *Třída jazyků* se nazývá *abstraktní*, jestliže obsahuje nějaký neprázdný jazyk a je uzavřena vůči

1. sjednocení,
2. průniku s regulárním jazykem,
3. pozitivní iteraci (tj. operaci $^+$),
4. nevypouštějícímu homomorfismu,
5. inverznímu homomorfismu.

Abstraktní třída jazyků je plná, jestliže je uzavřena nejen vůči nevypouštějícímu, ale vůči libovolnému homomorfismu.

Třídy jazyků typu 0, bezkontextových jazyků a regulárních jazyků tvoří plné abstraktní třídy jazyků. Kontextové jazyky jsou příkladem abstraktní třídy jazyků, která není plná.

Pro ilustraci uvedeme několik uzávěrových vlastností plných abstraktních tříd jazyků.

Věta 4.8.16. *Každá plná abstraktní třída jazyků je uzavřena vůči*:

a) *zřetězení jazyků*,

b) *operaci iterace*,

c) *regulární substituci* [*tj. takové substituci*

$$\sigma\colon \Sigma^* \to \mathscr{P}(\Delta^*),$$

u níž pro každé $a \in \Sigma$ je $\sigma(a)$ *regulární jazyk*],

d) *sekvenčnímu zobrazení*,

e) *inverznímu sekvenčnímu zobrazení*,

f) *levému i pravému kvocientu s regulárním jazykem.*

Důkaz této věty i další informace o abstraktních třídách jazyků je možno nalézt např. v knize [28].

4.9. Normální formy, třídy generátorů

Stejnou službu, jakou v zjednodušování úvah o bezkontextových jazycích poskytuje Chomského normální forma, s níž jsme se seznámili v čl. 4.7, nabízí v jiných souvislostech některá z dalších normálních forem. Seznámíme se proto stručně ještě se dvěma dalšími důležitými normálními formami bezkontextových gramatik, Greibachové normální formou a tzv. operátorovými gramatikami.

Definice 4.9.1. Bezkontextová gramatika je v *Greibachové normální formě* (dále jen GNF), právě když všechna pravidla jsou tvaru $A \to a\alpha$, kde A je neterminál, a je terminál a α je řetěz neterminálů (popřípadě prázdný).

Věta 4.9.2. *Ke každému bezkontextovému jazyku* L *existuje gramatika* $\mathscr{G}$ *v Greibachové normální formě taková, že*

$$L(\mathscr{G}) = L - \{e\}.$$

Důkaz věty, tzn. i konstrukci, kterou se libovolná nevypouštějící bezkontextová gramatika převádí na ekvivalentní gramatiku v GNF, najde čtenář v knize [20].

Poznámka 4.9.3. Je užitečné vědět, že každou gramatiku v GNF lze dále zjednodušit na ekvivalentní gramatiku v GNF, která má pouze pravidla tvaru $A \to a\alpha$, kde A je neterminál, a terminál a α řetěz neterminálů délky maximálně 2.

Jestliže totiž $\mathscr{G} = (\Pi, \Sigma, S, P)$ je libovolná gramatika v GNF, jejíž nejdelší pravá strana pravidla obsahuje $n > 2$ neterminálů, lze $\mathscr{G}$ převést na ekvivalentní gramatiku

$$\mathscr{G}' = (\Pi', \Sigma, S, P')$$

v GNF, u níž pravá strana žádného pravidla neobsahuje více než $n - 1$ neterminálů. Konstrukce vypadá takto:

$$\Pi' = \Pi \cup \Pi^2$$

a P' se sestrojí podle následujícího předpisu.

Každé pravidlo typu $X \to a^{\vee}_1 \ldots Y_n$ se nahradí pravidlem

$$X \to aY_1 \ldots Y_{n-2}(Y_{n-1}, Y_n).$$

Dále se do P' přidají pravidla, na jejichž levé straně je neterminál z Π^2:

a) $(X_1, X_2) \to aY_1 \ldots Y_k X_2$,

jestliže $k < n - 1$ a $X_1 \to aY_1 \ldots Y_k \in P$,

b) $(X_1, X_2) \to aY_1 \ldots Y_{n-2}(Y_{n-1}, X_2)$,

jestliže $X_1 \to aY_1 \ldots Y_{n-1} \in P$,

c) $(X_1, X_2) \to aY_1 \ldots Y_{n-3}(Y_{n-2}, Y_{n-1})(Y_n, X_2)$,

jestliže $X_1 \to aY_1 \ldots Y_n \in P$.

(Podrobněji v knize [17].)

Definice 4.9.4. Bezkontextová gramatika $\mathscr{G} = (\Pi, \Sigma, S, P)$ se nazývá *operátorová gramatika*, jestliže se uvnitř pravé strany

žádného z pravidel nevyskytují dva sousedící neterminály. [Tzn. gramatika má pravidla tvaru

$$A \to \alpha,$$

kde

$$A \in \Pi,\ \alpha \in (\Pi \cup \Sigma)^* - (\Pi \cup \Sigma)^* \Pi^2 (\Pi \cup \Sigma)^*.]$$

Věta 4.9.5. *Ke každému bezkontextovému jazyku L existuje operátorová gramatika $\mathscr{G}$ tak, že $L(\mathscr{G}) = L$.*

Důkaz. Budiž $\mathscr{G}_1 = (\Pi_1, \Sigma, S, P_1)$ gramatika v GNF generující $L - \{e\}$. Podle 4.9.3 můžeme předpokládat, že $\mathscr{G}_1$ má pouze pravidla tvaru

$$X \to a,\ X \to aY,\ X \to aYZ,$$

kde

$$a \in \Sigma,\ X, Y, Z \in \Pi_1.$$

Sestrojíme

$$\mathscr{G} = (\Pi, \Sigma, S, P),$$

kde

$$\Pi = \Pi_1 \cup \Sigma \times \Pi_1$$

a P je vytvořeno takto:

1. z P se převezmou pravidla tvaru $X \to a$, $X \to aY$,
2. místo každého pravidla tvaru $X \to aYZ$ se přidá soustava pravidel $X \to aY(b, Z)$ pro všechna $b \in \Sigma$,
3. přidají se pravidla

$$(b, Z) \to e,$$

jestliže $Z \to b \in P_1$,

$$(b, Z) \to A,$$

jestliže $Z \to bA \in P_1$,

$$(b, Z) \to Ac(c, B) \text{ pro všechna } c \in \Sigma,$$

jestliže $Z \to bAB \in P_1$.

V případě, že $e \in L$, se přidá nový počáteční neterminál S' a pravidla $S' \to S \mid e$.

Jinou možnost, jak zjednodušit úvahy o bezkontextových jazycích, nabízí charakterizace této třídy pomocí množin generátorů. Každá taková množina je podtřídou bezkontextových jazyků, z níž lze každý bezkontextový jazyk získat pomocí operací, vůči nimž je celá třída bezkontextových jazyků uzavřena. Nejznámějším příkladem je množina tzv. Dyckových jazyků.

Definice 4.9.6. Budiž

$$\Sigma_n = \{a_1, a'_1, a_2, a'_2, \ldots, a_n, a'_n\}$$

abeceda, $n \geqq 1$. *Dyckův jazyk* D_n *nad abecedou* Σ_n je jazyk generovaný gramatikou

$$S \to e \mid SS \mid a_1 S a'_1 \mid \ldots \mid a_n S a'_n .$$

Jestliže tedy např.

$$\Sigma_2 = \{(,), [,]\},$$

potom do Dyckova jazyka D_2 patří slova

$$([]())[])(()[]), \quad (())[()][[]()]$$

apod.

Věta 4.9.7. *Ke každému bezkontextovému jazyku L existuje regulární jazyk R a homomorfismus h tak, že*

$$L = h(D \cap R)$$

pro nějaký Dyckův jazyk D.

Důkaz je možno nalézt např. v knihách [17], [28].

Zajímavým zjištěním je skutečnost, že celou třídu bezkontextových jazyků je možné získat z jediného bezkontextového jazyka pomocí jediného typu operace.

Věta 4.9.8. *Existuje bezkontextový jazyk* L_0 *takový, že ke každému bezkontextovému jazyku L existuje homomorfismus h tak, že*

$$L - \{e\} = h^{-1}(L_0).$$

Důkaz je obsažen v knize [17].

5. ZÁKLADNÍ METODY SYNTAKTICKÉ ANALÝZY

ZÁKLADNÍ ČÁST

5.1. LL(1) gramatiky

V této kapitole se podrobněji vrátíme k tématu čl. 4.3 a 4.6, ve kterých jsme se v hrubých rysech seznámili s principy syntaktické analýzy shora a zdola. Při výkladu jsme si pomáhali odkazem na jakýsi „dodatečný zdroj informací", který řídil volbu vhodného pokračování. Nyní se soustředíme na způsob, jakým se tyto dodatečné informace získávají v prakticky užívaných metodách. Začneme jednoduchým příkladem.

Příklad 5.1.1. Gramatika $\mathscr{G}$:

(1) $S \to aSa$,

(2) $S \to bSb$,

(3) $S \to c$

generuje jazyk

$$\{wcw^{\mathrm{R}};\ w \in \{a, b\}^*\}.$$

Analýzu řetězu

$$abcba \in L(\mathscr{G})$$

metodou zavedenou v čl. 4.3 popíšeme touto tabulkou:

Krok výpočtu	Obsah zásobníku	Prováděná operace	Čtený vstupní symbol	Číslo pravidla
1	*S*	přepsání *S* na *aSa*	–	1
2	*aSa*	krácení	*a*	–
3	*Sa*	přepsání *S* na *bSb*	–	2
4	*bSba*	krácení	*b*	–
5	*Sba*	přepsání *S* na *c*	–	3
6	*cba*	krácení	*c*	–
7	*ba*	krácení	*b*	–
8	*a*	krácení	*a*	–
9	–	slovo přijato		

Tab. 9

V krocích 1, 3 a 5 popsaného výpočtu (tzn. vždy při přepisování neterminálu *S*) bylo třeba vybrat jedno z několika možných pokračování. Jaký doplňující zdroj informací nám daná gramatika pro toto rozhodování nabízí? V našem konkrétním případě je odpověď jednoduchá. Po použití pravidla 1 se na vrcholu zásobníku objeví symbol *a*, který se v následujícím kroku musí krátit proti stejnému symbolu na vstupu. Přepsat *S* podle pravidla 1 má tedy naději na úspěšné pokračování výpočtu pouze tehdy, jestliže následující vstupní symbol je *a*. Podobně použití pravidla 2 připadá v úvahu pouze následuje-li na vstupu symbol *b* (jako v kroku 3 výpočtu) a použití pravidla 3 pouze následuje-li *c*. Je vidět, že v daném případě lze podle následujícího vstupního symbolu eliminovat všechna možná pokračování až na jedno. Doplňující informace potřebná pro deterministický průběh analýzy je tedy obsažena v bezprostředně následujícím úseku vstupního řetězu. Algoritmus analýzy dané gramatiky je možné vyjádřit ve formě tabulky takto:

		následující vstupní symbol			
		a	*b*	*c*	*e*
vrchní symbol zásobníku	*S*	*aSa*; 1	*bSb*; 2	*c*; 3	**chyba**
	a	**krátit**	**chyba**	**chyba**	**chyba**
	b	**chyba**	**krátit**	**chyba**	**chyba**
	c	**chyba**	**chyba**	**krátit**	**chyba**
	e	**chyba**	**chyba**	**chyba**	**přijmout**

Tab. 10

Tabulka určuje operaci, která se má provést, v závislosti na následujícím vstupu (*e* v případě, že slovo je přečteno) a na vrchním symbolu zásobníku (*e* zde znamená prázdný zásobník).

Operace jsou těchto čtyř různých typů:

1. Nahrazení vrchního symbolu v zásobníku řetězem (pravou stranou *i*-tého pravidla) a vytištění výstupního symbolu (*i*) (což je v tabulce zachyceno uvedením těchto dvou údajů).

2. Krácení – odstranění vrchního symbolu v zásobníku a posun čtecí hlavy o jedno políčko doprava.

3. Chyba – ukončení výpočtu znamenající, že analyzované slovo nepatří do $L(\mathscr{G})$.

4. Přijetí – ukončení výpočtu znamenající, že analyzované slovo patří do $L(\mathscr{G})$.

Není těžké vymezit vlastnosti gramatiky, o které jsme se v předchozím případě opírali. Jedná se o vlastnosti charakterizující třídu tzv. jednoduchých LL(1) gramatik.

Definice 5.1.2. Bezkontextová gramatika $\mathscr{G} = (\Pi, \Sigma, S, P)$ je *jednoduchá* LL(1) *gramatika*, jestliže splňuje tyto dvě vlastnosti:

1. u každého pravidla začíná řetěz na jeho pravé straně terminálem;

2. jestliže dvě pravidla mají stejnou levou stranu, potom se liší v terminálu, kterým začínají jejich pravé strany.

Poznámka 5.1.3. Označení LL(1) je přejato z anglického názvosloví. Jednoduché LL(1) gramatiky jsou speciálním případem

tzv. LL(k) gramatik, se kterými se seznámíme v čl. 5.6. Symbol LL(k) označuje, že se jedná o gramatiku umožňující analýzu, při níž je analyzované slovo čteno zleva doprava (left-to-right) a je vydáván tzv. levý rozbor (left parse); analýza se v každém kroku opírá o prohlížení následujících k vstupních symbolů.

Příklad 5.1.4. 1. Gramatika

$$S \to aSbS \mid bA\,,\quad A \to bA \mid a$$

je jednoduchá LL(1) gramatika.

2. Gramatika

$$S \to aSbS \mid aA\,,\quad A \to bA \mid a$$

není jednoduchá LL(1) gramatika.

3. Gramatika

$$S \to aSbS \mid Ab\,,\quad A \to bA \mid a$$

není jednoduchá LL(1) gramatika.

Připomeňme, že cílem syntaktické analýzy shora dolů je nejen zjistit, zda daný řetěz lze odvodit podle příslušné gramatiky, ale udat i příslušnou levou derivaci, přesněji tzv. levý rozbor.

Definice 5.1.5. Budiž $\mathscr{G} = (\Pi, \Sigma, S, P)$ bezkontextová gramatika, jejíž pravidla jsou očíslována, $w \in L(\mathscr{G})$ a

$$S \overset{i_1}{\Rightarrow} \alpha_1 \overset{i_2}{\Rightarrow} \ldots \overset{i_k}{\Rightarrow} \alpha_k = w$$

(nějaká) jeho levá derivace. (Čísla i_j u šipek udávají číslo pravidla, které je v příslušném kroku použito.) Potom posloupnost $i_1, \ldots, i_k$ nazveme *levým rozborem řetězu w podle gramatiky* $\mathscr{G}$.

Poznámka 5.1.6. Až do konce kapitoly budeme předpokládat, že každá bezkontextová gramatika, s níž budeme pracovat, má očíslovaná pravidla.

Příklad 5.1.7. Levým rozborem řetězů *abacaba*, *bbabcbabb* podle gramatiky z př. 5.1.1 jsou po řadě posloupnosti 1, 2, 1, 3 a 2, 2, 1, 2, 3.

Pro jednoduché LL(1) gramatiky budeme algoritmy analýzy zadávat ve formě tzv. LL(1) analyzátorů.

Definice 5.1.8. Budiž $\mathscr{G} = (\Pi, \Sigma, S, P)$ bezkontextová gramatika. Potom LL(1) *analyzátorem* vztahujícím se ke $\mathscr{G}$ nazveme každou funkci

$$\mathscr{M}: (\Pi \cup \Sigma \cup \{e\}) \times (\Sigma \cup \{e\}) \to$$
$$\to \{\textbf{krátit}, \textbf{chyba}, \textbf{přijmout}\} \cup$$
$$\cup \{(\eta; i);\ \eta \text{ je pravá strana } i\text{-tého pravidla}\}$$

takovou, že

$$\mathscr{M}(a, a) = \textbf{krátit}$$

pro všechna $a \in \Sigma$,

$$\mathscr{M}(e, e) = \textbf{přijmout},$$

a jestliže

$$\mathscr{M}(X, a) = (\beta; i),$$

potom $X \to \beta$ je i-té pravidlo.

Příklad 5.1.9. Tabulka 10 v př. 5.1.1 udává jeden z LL(1) analyzátorů vztahujících se ke gramatice z téhož příkladu. Navíc je to právě ten analyzátor, který poskytuje správné výsledky při analýze. Zavedeme pojmy, jež nám umožní i tuto skutečnost přesněji vyjádřit.

Definice 5.1.10. *Situací* LL(1) *analyzátoru* $\mathscr{M}$ vztahujícího se ke gramatice $\mathscr{G} = (\Pi, \Sigma, S, P)$ nazveme jednak každou trojici (u, η, y), kde $u \in \Sigma^*$, $\eta \in (\Pi \cup \Sigma)^*$, y značí posloupnost čísel pravidel, jednak dvojice (**přijmout**, y), kde y značí posloupnost čísel pravidel, jednak symbol **chyba**.

Řekneme, že *analyzátor* $\mathscr{M}$ *bezprostředně přejde ze situace* E *do situace* E' (označíme $E \vdash_{\mathscr{M}} E'$), jestliže

1. $E = (au, a\eta, y)$

pro nějaké

$$a \in \Sigma \text{ a } E' = (u, \eta, y)$$

[$E \vdash_{\mathscr{M}} E'$ podle pravidla $\mathscr{M}(a, a) =$ **krátit**], nebo

2. $E = (au, X\eta, y)$

pro nějaké

$$a \in (\Sigma \cup \{e\}),\ X \in \Pi,\ \mathscr{M}(X, a) = (\beta, i)$$

a

$$E' = (au, \beta\eta, yi),$$

3. $E = (e, e, y)$ a $E' =$ (**přijmout**, y),

4. $E = (au, X\eta, y)$

pro nějaké

$$a \in (\Sigma \cup \{e\}),\ X \in \Pi \cup \Sigma \cup \{e\},$$

$$\mathscr{M}(X, a) = \textbf{chyba} \text{ a } E' = \textbf{chyba}.$$

Řekneme, že $\mathscr{M}$ *přejde ze situace* E *do* E' (označíme $E \vdash^*_{\mathscr{M}} E'$), jestliže existují $E_1, \ldots, E_n$, $n \geqq 1$, tak, že $E = E_1$, $E' = E_n$ a $E_i \vdash_{\mathscr{M}} E_{i+1}$ pro každé i, $1 \leqq i < n$.

Příklad 5.1.11. Analyzátor popsaný v př. 5.1.1 provádí nad slovem *abcba* výpočet

$$(abcba, S, e) \vdash (abcba, aSa, 1) \vdash (bcba, Sa, 1) \vdash$$

$$\vdash (bcba, bSba, 12) \vdash (cba, Sba, 12) \vdash (cba, cba, 123) \vdash$$

$$\vdash (ba, ba, 123) \vdash (a, a, 123) \vdash (e, e, 123) \vdash (\textbf{přijmout}, 123).$$

Výsledkem je konstatování, že předložený řetěz patří do $L(\mathscr{G})$ a navíc levý rozbor tohoto řetězu.

Nad slovem *baccab* proběhne výpočet

$$(baccab, S, e) \vdash (baccab, bSb, 2) \vdash (accab, Sb, 2) \vdash$$

$$\vdash (accab, aSab, 21) \vdash (ccab, Sab, 21) \vdash (ccab, cab, 213) \vdash$$

$$\vdash (cab, ab, 213) \vdash \textbf{chyba}.$$

Výsledkem je konstatování, že předložený řetěz nepatří do $L(\mathscr{G})$.

Definice 5.1.12. Nechť $\mathscr{M}$ je LL(1) analyzátor vztahující se k bezkontextové gramatice $\mathscr{G} = (\Pi, \Sigma, S, P)$. *Počáteční situací analyzátoru* $\mathscr{M}$ *nad slovem* $u \in \Sigma^*$ nazveme situaci (u, S, e). Řekneme, že $\mathscr{M}$ je LL(1) *analyzátorem gramatiky* $\mathscr{G}$, jestliže pro každé slovo $u \in \Sigma^*$ platí:

1. jestliže $u \notin L(\mathscr{G})$, potom $(u, S, e) \vdash_{\mathscr{M}}^* \textbf{chyba}$,
2. jestliže $u \in L(\mathscr{G})$, potom $(u, S, e) \vdash_{\mathscr{M}} (\textbf{přijmout}, y)$, kde y je nějaký levý rozbor slova u podle $\mathscr{G}$ (u jednoznačných gramatik, o které nám v dalším výkladu výhradně půjde, je y určen jednoznačně).

Věta 5.1.13. *Ke každé jednoduché* LL(1) *gramatice lze sestrojit její* LL(1) *analyzátor.*

Důkaz. Budiž $\mathscr{G} = (\Pi, \Sigma, S, P)$ jednoduchá LL(1) gramatika. Její LL(1) analyzátor sestrojíme následující konstrukcí.

Konstrukce 5.1.14 [konstrukce LL(1) analyzátoru pro jednoduchou LL(1) gramatiku].

Analyzátor $\mathscr{M}$ pro $\mathscr{G}$ definujme předpisem:

$$\mathscr{M}(a, a) = \textbf{krátit}$$

pro každé $a \in \Sigma$,

$$\mathscr{M}(e, e) = \textbf{přijmout},$$

$$\mathscr{M}(A, a) = (\beta, i)$$

pro $A \in \Pi$ a $a \in \Sigma$, jestliže i-té pravidlo z P je tvaru $A \to a\alpha$, kde $\alpha \in (\Sigma \cup \Pi)^*$ a $a\alpha = \beta$,

$$\mathscr{M}(X, a) = \textbf{chyba} \text{ ve všech ostatních případech}.$$

$$(X \in \Pi \cup \Sigma \cup \{e\},\ a \in \Sigma \cup \{e\}.)$$

Zbývá ověřit, že $\mathscr{M}$ je skutečně analyzátorem pro $\mathscr{G}$. To ale snadno nahlédneme, jakmile srovnáme analyzátor $\mathscr{M}$ s nedeterministickým zásobníkovým automatem $\mathscr{M}'$, který vzniká konstrukcí podle důkazu věty 4.5.1. Víme už, že $N(\mathscr{M}') = L(\mathscr{G})$. Všem

pravidlům typu $\mathscr{M}(X, a) = (\beta, i)$ odpovídá u $\mathscr{M}'$ při označení tam zavedeném $(q, \beta) \in \delta(q, e, X)$. Kromě toho situace typu (**přijmout**, y) je u $\mathscr{M}$ vždy předcházena situací, která u zásobníkového automatu odpovídá přijetí prázdným zásobníkem. Každému přijímajícímu výpočtu podle $\mathscr{M}$ proto odpovídá přijímající výpočet podle $\mathscr{M}'$.

Obráceně každému přijímajícímu výpočtu podle zásobníkového automatu $\mathscr{M}'$ odpovídá přijímající výpočet podle analyzátoru $\mathscr{M}$. Jestliže totiž $\mathscr{M}'$ použije pravidlo

$$(q, e, X) \xrightarrow{\delta} (q, \beta),$$

potom

a) v P je obsaženo pravidlo $X \to \beta$ s číslem, řekněme i;

b) β začíná jistým terminálem a [protože $\mathscr{G}$ je jednoduchá LL(1)];

c) protože se jedná o úspěšný výpočet, terminál a, který se takto ocitne na vrcholu zásobníku, bude v následujícím taktu krácen proti následujícímu vstupnímu symbolu, který je tedy také a.

Z toho je vidět, že v okamžiku, kdy $\mathscr{M}'$ užívá instrukci

$$(q, e, X) \xrightarrow{\delta} (q, \beta),$$

může $\mathscr{M}$ užít pravidlo

$$\mathscr{M}(X, a) = (\beta, i)$$

pro jisté i a a.

V pravidlech pro operaci krácení se $\mathscr{M}$ i $\mathscr{M}'$ shodují a situacím vedoucím u $\mathscr{M}$ k situaci **chyba** odpovídají u $\mathscr{M}'$ situace, pro něž není δ definováno. $\mathscr{M}$ proto přijímá právě slova z $N(\mathscr{M})$, tj. $L(\mathscr{G})$.

Nahrazování neterminálů na vrcholu zásobníku odpovídá levé derivaci, a proto jestliže

$$(u, S, e) \vdash^*_{\mathscr{M}} (\textbf{přijmout}, y),$$

je y levým rozborem u.

Příklad 5.1.15. Gramatika daná pravidly

(1) $S \to aSA$,

(2) $S \to bB$,

(3) $A \to aAb$,

(4) $A \to b$,

(5) $B \to aA$

je jednoduchá LL(1) gramatika. LL(1) analyzátor pro tuto gramatiku získaný podle konstrukce 5.1.14 je dán tabulkou

	a	b	e
S	aSA; 1	bB; 2	
A	aAb; 3	b; 4	
B	aA; 5		
a	**krátit**		
b		**krátit**	
e			**přijmout**

Tab. 11

Nevyplněným políčkům tabulky odpovídá položka **chyba**.

LL(1) analyzátor pro jednoduché LL(1) gramatiky se nám podařilo sestrojit díky tomu, že při přepisování neterminálu na vrcholu zásobníku podle dvou různých pravidel se v následujícím kroku objeví na vrcholu v každém z obou případů různý terminál. Je proto možné připustit pouze to přepsání, u něhož se tento terminál bude shodovat s následujícím vstupním symbolem. Pro konstrukci ovšem není podstatné, že se rozlišující terminál objeví na vrcholu zásobníku hned v následujícím kroku.

Díky tomu je možné ji použít i u gramatik, které takové rozlišení pravidel pomocí vzniklých terminálů dovolí popřípadě až po více krocích. To nás vede k vymezení pojmu LL(1) gramatik.

Definice 5.1.16. Pro každou bezkontextovou gramatiku

$$\mathcal{G} = (\Pi, \Sigma, S, P),$$

přirozené číslo $k \geqq 1$ a $\omega \in (\Pi \cup \Sigma)^*$ definujeme množinu slov

$$k_{\mathcal{G}}(\omega) = \{w \in \Sigma^*;\ \text{buď } |w| < k \text{ a } \omega \Rightarrow_{\mathcal{G}}^* w,$$

$$\text{nebo } |w| = k \text{ a } \omega \Rightarrow_{\mathcal{G}}^* wx \text{ pro nějaké } x \in \Sigma^*\}.$$

Množinu slov $k_{\mathcal{G}}(\omega)$ dostaneme tedy tak, že ze všech terminálních slov, která lze odvodit z ω, vezmeme vždy prvních k symbolů (popřípadě celé slovo, je-li kratší než k).

Přirozeným způsobem definujeme rozšíření na množiny slov: pro libovolné $L \subseteq (\Pi \cup \Sigma)^*$ položíme

$$k_{\mathcal{G}}(L) = \{k_{\mathcal{G}}(\omega);\ \omega \in L\}.$$

Příklad 5.1.17. Označme symbolem $\mathcal{G}$ gramatiku

$$S \to AaaS \mid bAb \mid e,$$

$$A \to c \mid d \mid e.$$

Potom např.

$$1_{\mathcal{G}}(Aa) = \{c, d, a\},\ 1_{\mathcal{G}}(S) = \{e, a, b, c, d\}.$$

V případě, že nebude hrozit nedorozumění, budeme index $\mathcal{G}$ vynechávat a psát pouze $k(\omega)$.

Všimněte si, že pro terminální řetěz u je množina $k(u)$ vždy jednobodová — obsahuje počáteční úsek u délky k.

Z toho, co jsme zatím uvedli, je zřejmé, že analogicky jako pro jednoduché LL(1) gramatiky lze LL(1) analyzátor sestrojit i pro gramatiky, které mají tyto dvě vlastnosti:

a) pro každé pravidlo $A \to \eta$ je $e \notin 1(\eta)$ (odtud mj. plyne, že gramatika je nevypouštějící);

b) pro každá dvě různá pravidla $A \to \eta$, $A \to \xi$ je

$$1(\eta) \cap 1(\xi) = \emptyset.$$

Přepsání neterminálu na vrcholu zásobníku podle dvou různých pravidel takové gramatiky se díky a) a b) odliší v okamžiku, kdy se na vrcholu zásobníku objeví terminál. Protože tento terminál má souhlasit s následujícím vstupním symbolem, je podle tohoto vstupního symbolu možné určit, podle kterého pravidla se má neterminál přepsat. Takže např. pravidlo $A \to \eta$ je možné použít, pouze když následující vstupní symbol patří do $1(\eta)$.

U gramatik, pro které neplatí podmínka a), je situace komplikovanější, protože vrchní symbol v zásobníku, řekněme A, se může postupně přepsat na prázdné slovo a první terminál, který se pak objeví na vrcholu zásobníku, je vygenerován z řetězu ležícího v zásobníku pod A.

Následující definice vymezují gramatiky, u nichž ani neplatnost podmínky a) není překážkou pro konstrukci jejich LL(1) analyzátoru.

Definice 5.1.18. Budiž $\mathscr{G} = (\Pi, \Sigma, S, P)$ bezkontextová gramatika a $\omega \in (\Pi \cup \Sigma)^*$. Řekneme, že ω je *levou větnou formou podle gramatiky* $\mathscr{G}$, jestliže existuje levá derivace $S \Rightarrow_{\mathscr{G}}^* \omega$.

Definice 5.1.19. Bezkontextová gramatika

$$\mathscr{G} = (\Pi, \Sigma, S, P)$$

se nazývá (*silná*) LL(1) *gramatika*, jestliže pro každá její dvě pravidla $A \to \alpha$, $A \to \beta$ taková, že $\alpha \neq \beta$, a pro každé dvě levé větné formy $uA\eta$, $vA\xi$ [kde $u, v \in \Sigma^*$, $\eta, \xi \in (\Sigma \cup \Pi)^*$] platí:

$$1(\alpha\eta) \cap 1(\beta\xi) = \emptyset .$$

Poznámka 5.1.20. V čl. 5.6 budeme definovat LL(k) gramatiky a silné LL(k) gramatiky a ukážeme, že pro $k \geqq 2$ se oba pojmy liší, zatímco pro $k = 1$ splývají. Z toho důvodu předchozí definice vymezuje jak LL(1) gramatiky, tak silné LL(1) gramatiky.

Příklad 5.1.21. 1. Gramatika $\mathscr{G}$ daná pravidly

(1) $A \to BA'$,

(2) $A' \to +BA'$,

(3) $A' \to e$,

(4) $B \to CB'$,

(5) $B' \to *CB'$,

(6) $B' \to e$,

(7) $C \to (A)$,

(8) $C \to a$

s počátečním neterminálem A je LL(1) gramatika, jak může čtenář sám ověřit. Ověřit platnost podmínky z 5.1.19 stačí v tomto případě pro A'-pravidla, B'-pravidla a C-pravidla.

Pro ilustraci ukážeme, že podmínka platí pro B'-pravidla.

Pro všechny levé větné formy $uB'\eta$ a $vB'\xi$ je

$$1(*CB'\eta) = \{*\} \text{ a } 1(e\xi) = 1(\xi).$$

Je tedy třeba prozkoumat, jak vypadá množina $1(\xi)$ pro každé ξ takové, že $vB'\xi$ je levá větná forma pro vhodné $v \in \Sigma^*$. Symbol B' se v takové levé větné formě může objevit aplikací pravidel (4) a (5). Pravidlo (5) muselo být použito na nějakou větnou formu tvaru $v_1B'\xi$, pravidlo (4) na větnou formu tvaru $v_2B\xi$ (kde $v_1 * C \Rightarrow^* v$ a $v_2C \Rightarrow^* v$). První případ nepřináší o řetězu ξ žádnou novou informaci, stačí se omezit na případ druhý. Ve větné formě $v_2B\xi$ se B může objevit buď použitím pravidla (1), nebo (2). V obou případech to znamená, že ξ je tvaru $\xi = A'\xi'$ pro nějaké ξ'. Odtud dále plyne, že

$$(*) \qquad 1(\xi) = 1(A'\xi') = \{+\} \cup 1(\xi'),$$

neboť A' se může dále přepisovat buď pomocí (2), nebo (3). Dostali jsme se tak ke zkoumání ξ', kde $v_3A\xi'$ je nějaká levá větná forma. Zde jsou opět dvě možnosti:

1. $v_3A\xi' = A$, což odpovídá začátku derivace. V tom případě $\xi' = e$.

2. A vzniklo aplikací pravidla (7). Pak $\xi' =)\xi''$ pro nějaké ξ''. Z toho vyplývá, že

$$1(\xi') = \{e,)\}.$$

Dostali jsme tak

$$1(\xi) = \{+,), e\}.$$

Jelikož

$$\{*\} \cap \{+,), e\} = \emptyset,$$

je podmínka pro B' splněna.

2. Gramatika $\mathscr{G}'$

(1) $A \to A + B,$

(2) $A \to B,$

(3) $B \to B * C,$

(4) $B \to C,$

(5) $C \to (A),$

(6) $C \to a$

není LL(1). Snadno se přesvědčíte, že např.

$$1(B * C) = \{(, a\} \text{ a } 1(C) = \{(, a\}.$$

Pro libovolné levé větné formy $uB\eta$ a $vB\xi$ je proto

$$1(B * C\eta) \cap 1(C\xi) = \{(, a\} \neq \emptyset.$$

Všimněte si, že přitom $L(\mathscr{G}') = L(\mathscr{G})$.

Věta 5.1.22. *Ke každé* LL(1) *gramatice lze sestrojit její* LL(1) *analyzátor.*

Důkaz. Budiž $\mathscr{G} = (\Pi, \Sigma, S, P)$ LL(1) gramatika. Její LL(1) analyzátor sestrojíme následující konstrukcí:

Konstrukce 5.1.23 (konstrukce LL(1) analyzátoru pro LL(1) gramatiku).

Analyzátor $\mathscr{M}$ definujeme tímto předpisem:

$$\mathscr{M}(a, a) = \textbf{krátit}$$

pro každé $a \in \Sigma$,

$$\mathscr{M}(e, e) = \textbf{přijmout},$$

$$\mathscr{M}(A, a) = (\beta, i)$$

pro $A \in \Pi$, $a \in \Sigma \cup \{e\}$, jakmile $A \to \beta$ je i-té pravidlo P a existuje levá větná forma $uA\eta$ taková, že $a \in 1(\beta\eta)$,

$$\mathscr{M}(X, a) = \textbf{chyba} \text{ ve všech ostatních případech.}$$

Otázku, jak efektivně testovat, zda pro dané pravidlo $A \to \beta$ a

$$a \in \Sigma \cup \{e\}$$

existuje levá větná forma $uA\eta$ tak, že

$$a \in 1(\beta\eta),$$

zodpovíme až v čl. 5.6.

Ověření, že $\mathscr{M}$ je LL(1) analyzátorem pro $\mathscr{G}$, je analogické jako v důkazu 5.1.13.

Příklad 5.1.24. LL(1) analyzátor pro gramatiku $\mathscr{G}$ z př. 5.1.21 je dán touto tabulkou 12:

	a	(	)	+	*	e
A	BA'; 1	BA'; 1				
A'			e; 3	$+BA'$; 2		e; 3
B	CB'; 4	CB'; 4				
B'			e; 6	e; 6	$*CB'$; 5	e; 6
C	a; 8	(A); 7				
a	**krátit**					
(		**krátit**				
)			**krátit**			
+				**krátit**		
*					**krátit**	
e						**přijmout**

Tab. 12

Proberme si výpočet analyzátoru nad řetězem $a + a * a$:

$$(a + a * a, A, e) \vdash (a + a * a, BA', 1) \vdash$$
$$\vdash (a + a * a, CB'A', 14) \vdash (a + a * a, aB'A', 148) \vdash$$
$$\vdash (+a * a, B'A', 148) \vdash (+a * a, A', 1486) \vdash$$
$$\vdash (+a * a, +BA', 14862) \vdash (a * a, BA', 14862) \vdash$$
$$\vdash (a * a, CB'A', 148624) \vdash (a * a, aB'A', 1486248) \vdash$$
$$\vdash (*a, B'A', 1486248) \vdash (*a, *CB'A', 14862485) \vdash$$
$$\vdash (a, CB'A', 14862485) \vdash (a, aB'A', 148624858) \vdash$$
$$\vdash (e, B'A', 148624858) \vdash (e, A', 1486248586) \vdash$$
$$\vdash (e, e, 14862485863) \vdash (\textbf{přijmout}, 14862485863).$$

Analyzovaný řetěz tedy patří do $L(\mathscr{G})$ a 14862485863 je jeho levý rozbor.

Výpočet nad řetězem $a(a + a)$ vypadá takto:

$$(a(a + a), A, e) \vdash (a(a + a), BA', 1) \vdash$$
$$\vdash (a(a + a), CB'A', 14) \vdash (a(a + a), aB'A', 148) \vdash$$
$$\vdash ((a + a), B'A', 148) \vdash \textbf{chyba}.$$

Tento řetěz do $L(\mathscr{G})$ nepatří.

5.2. LR(0) gramatiky

LL(1) gramatiky v předchozím článku byly příkladem gramatik, u kterých lze deterministicky provádět analýzu shora. Pro analýzu zdola stejnou službu prokáží tzv. LR(0) gramatiky.

V článku 4.6 jsme podali rámcový návod pro analýzu zdola, ale ponechali jsme stranou otázku, jakým způsobem rozhodnout, které z možných pokračování je třeba vybrat. Jedná se o tato rozhodování:

1. Zda přesunout další vstupní symbol do zásobníku, nebo redukovat zásobník.

2. V případě, že zvolíme možnost redukovat zásobník, je ještě třeba rozhodnout, podle kterého pravidla výchozí gramatiky se má redukce provádět.

Příklad 5.2.1. Máme analyzovat řetěz $[[aa]a]$ generovaný gramatikou $\mathscr{G}$:

(1) $S' \to A$,

(2) $A \to [S]$,

(3) $A \to a$,

(4) $S \to SA$,

(5) $S \to A$.

Neterminály jsou S', A, S, terminály a,], [, počáteční symbol je S'. Průběh analýzy je zachycen v tab. 13. Při zápisu obsahu zásobníku nyní stejně jako v čl. 4.6 odpovídá vrcholu zásobníku pravý konec slova.

Podle tabulky skončí analýza po přečtení celého slova se zásobníkem obsahujícím pouze počáteční symbol, tzn. že slovo je

Číslo kroku	Obsah zásobníku	Zbývající úsek vstupního slova	Typ provedené operace	Číslo užitého pravidla
0	–	$[[aa]a]$	–	–
1	$[$	$[aa]a]$	přesun	–
2	$[[$	$aa]a]$	přesun	–
3	$[[a$	$a]a]$	přesun	–
4	$[[A$	$a]a]$	redukce	3
5	$[[S$	$a]a]$	redukce	5
6	$[[Sa$	$]a]$	přesun	–
7	$[[SA$	$]a]$	redukce	3
8	$[[S$	$]a]$	redukce	4
9	$[[S]$	$a]$	přesun	–
10	$[A$	$a]$	redukce	2
11	$[S$	$a]$	redukce	5
12	$[Sa$	$]$	přesun	–
13	$[SA$	$]$	redukce	3
14	$[S$	$]$	redukce	4
15	$[S]$	–	přesun	–
16	A	–	redukce	2
17	S'	–	redukce	1

Tab. 13

přijato. Na výstupu je postupně vydávána posloupnost 3534253421, což je posloupnost čísel pravidel odpovídající pravé derivaci prováděné pozpátku.

Definice 5.2.2. Nechť $\mathscr{G} = (\Pi, \Sigma, S, P)$ je bezkontextová gramatika a

$$S \overset{i_1}{\Rightarrow} \alpha_1 \overset{i_2}{\Rightarrow} \alpha_2 \Rightarrow \ldots \overset{i_n}{\Rightarrow} \alpha_n = w$$

pravá derivace slova $w \in \Sigma^*$. Posloupnost $i_n, \ldots, i_2, i_1$ nazveme *pravým rozborem slova w*.

Výsledkem analýzy zdola je tedy rozhodnutí, zda slovo patří do daného jazyka, a v kladném případě pak také pravá derivace slova. Ukážeme si nyní, do jaké míry lze v každém kroku analýzy zúžit volbu možných pokračování na základě už provedených kroků.

Definice 5.2.3. Nechť $\mathscr{G} = (\Pi, \Sigma, S, P)$ je bezkontextová gramatika. *Položkou gramatiky* $\mathscr{G}$ nazveme každý výraz typu

$$X \to \alpha \,.\, \beta,$$

kde $X \to \alpha\beta$ je pravidlo gramatiky $\mathscr{G}$. Speciálně je $X \to .$ položkou, pokud $X \to e \in P$.

Pro každé $X \to \gamma \in P$ nazýváme $X \to \gamma.$ *úplnou položkou.*

Příklad 5.2.4. Sestrojme všechny položky gramatiky z př. 5.2.1:

$$S' \to .A\,,\ A \to .[S]\,,\ A \to .a\,,\ S \to .SA\,,\ S \to .A\,,$$

$$S' \to A.\,,\ A \to [.S]\,,\ A \to a.\,,\ S \to S.A\,,\ S \to A.\,,$$

$$A \to [S.]\,, \qquad S \to SA.\,,$$

$$A \to [S].\,.$$

Úplnými položkami jsou

$$S' \to A.\,,\ A \to [S].\,,\ A \to a.\,,\ S \to SA.\,,\ S \to A.\,.$$

Definice 5.2.5. Nechť $\mathscr{G} = (\Pi, \Sigma, S, P)$ je bezkontextová gramatika a $\omega \in (\Pi \cup \Sigma)^*$. Řekneme, že ω je *pravou větnou formou podle gramatiky* $\mathscr{G}$, jestliže existuje pravá derivace $S \Rightarrow_{\mathscr{G}}^* \omega$.

Definice 5.2.6. Nechť $\mathcal{G} = (\Pi, \Sigma, S, P)$ je bezkontextová gramatika, $\omega \in (\Sigma \cup \Pi)^*$ a $A \to \alpha . \beta$ položka. Řekneme, že $A \to \alpha . \beta$ je *platnou položkou pro řetěz* ω, jestliže existuje pravá větná forma $\eta A u$ $(u \in \Sigma^*)$ taková, že $\eta\alpha = \omega$.

Příklad 5.2.7. V případě gramatiky $\mathcal{G}$ z př. 5.2.1 je $S \to S.A$ platnou položkou např. pro $[[S$, neboť

$$S' \Rightarrow^* [[S]\, a] \Rightarrow [[SA]\, a]$$

je pravá derivace podle $\mathcal{G}$. Podobně např. $A \to a.$ je platná položka pro $[a$, zatímco k $[[a]\,[a$ neexistuje žádná platná položka.

Zjistit všechny platné položky pro určitý řetěz je možné konečným automatem, jak nyní dokážeme. Opřeme se přitom o několik lemmat platících pro libovolnou bezkontextovou gramatiku $\mathcal{G} = (\Pi, \Sigma, S, P)$. Pro množinu všech platných položek pro řetěz γ v nich budeme užívat označení $I(\gamma)$.

Lemma 5.2.8. *Jestliže* $A \to \alpha . B\beta \in I(\gamma)$, *potom také pro každé pravidlo* $B \to \vartheta$ *je* $B \to .\vartheta \in I(\gamma)$.

Důkaz. Jelikož $A \to \alpha . B\beta$ je platná položka pro γ, existuje pravá větná forma $\delta A w$ taková, že $\delta\alpha = \gamma$. Z pravé větné formy lze aplikací pravidla $A \to \alpha B \beta$ odvodit pravou větnou formu $\delta\alpha B\beta w$ a z ní pravou větnou formu $\delta\alpha B v w$, kde $\beta \Rightarrow^* v$. Z toho už přímo plyne, že

$$B \to .\vartheta \in I(\delta\alpha) = I(\gamma).$$

Lemma 5.2.9. *Jestliže*

$$B \to .\beta \in I(\gamma X), \quad X \in \Pi \cup \Sigma,$$

potom v $I(\gamma X)$ *existuje položka tvaru* $C \to \alpha X . \delta$ *pro nějaké*

$$C \in \Pi, \alpha, \delta \in (\Pi \cup \Sigma)^*$$

tak, že existuje pravé odvození $\delta \Rightarrow^* Bz$ *pro jisté* $z \in \Sigma^*$.

Důkaz. Podle předpokladu existuje pravá větná forma $\eta B u$ taková, že $\eta = \gamma X$, tzn. $\gamma X B u$ je pravá větná forma. Znak X vznikl v některém kroku pravého odvození řetězu $\gamma X B u$. Od vzniku X až do odvození $\gamma X B u$ nezasáhla pravá derivace do

úseku γX. Jinými slovy, existuje pravá větná forma ϱCv a pravidlo $C \to \alpha X \delta$ takové, že $\varrho \alpha X = \gamma X$ (a $\delta v \Rightarrow_p^* Bu$). Je tedy $C \to \alpha X \,.\, \delta \in I(\gamma X)$.

Lemma 5.2.10. *Jestliže* $B \to \alpha \,.\, \beta \in I(\gamma X)$ *a* $\alpha \neq e$, *potom* $\alpha = \alpha' X$ *pro nějaké* α' *a* $B \to \alpha' \,.\, X\beta \in I(\gamma)$.

Důkaz. Z předpokladu lemmatu plyne, že existuje pravá větná forma ηBu taková, že $\eta\alpha = \gamma X$. Jelikož je $\alpha \neq e$, znamená to, že $\alpha = \alpha' X$ pro nějaké α'. Tedy $\eta\alpha' = \gamma$, a to už dává

$$B \to \alpha' \,.\, X\beta \in I(\gamma)\,.$$

Důsledek 5.2.11. *Jestliže* $I(\gamma X) \neq \emptyset$, *potom* $I(\gamma) \neq \emptyset$.

Důkaz. Tvrzení plyne okamžitě z předchozích dvou lemmat, neboť pro každou položku $X \to \eta \,.\, \xi$ je buď $\eta = e$, nebo $\eta \neq e$.

Lemma 5.2.12. *Jestliže* $A \to \alpha \,.\, X\beta \in I(\gamma)$, $X \in \Pi \cup \Sigma$, *potom* $A \to \alpha X \,.\, \beta \in I(\gamma X)$.

Důkaz. Z předpokladu plyne existence pravé větné formy ηAu takové, že $\eta\alpha = \gamma$. Protože ale $\eta\alpha X = \gamma X$, plyne z toho okamžitě, že $A \to \alpha X \,.\, \beta \in I(\gamma X)$.

Lemma 5.2.13. *Pro libovolné* $\gamma \in (\Pi \cup \Sigma)^*$ *a* $X \in (\Pi \cup \Sigma)$ *lze množinu* $I(\gamma X)$ *určit na základě* $I(\gamma)$ *takto*: $I(\gamma X)$ *je nejmenší množina* M *taková, že*

1. $M \supseteq \{Y \to \alpha X \,.\, \beta;\ Y \to \alpha \,.\, X\beta \in I(\gamma)\}$,
2. *jestliže* $A \to \alpha \,.\, B\beta \in M$, *potom pro každé pravidlo* $B \to \vartheta \in P$ *je také* $B \to .\vartheta \in M$.

Důkaz. a) Podle lemmat 5.2.12 a 5.2.8 splňuje $I(\gamma X)$ podmínku 1 i 2.

b) Budiž M libovolná množina splňující podmínky 1 a 2. Zvolme libovolnou položku $B \to \alpha \,.\, \beta \in I(\gamma X)$. Rozeberme postupně případy $\alpha = e$ a $\alpha \neq e$.

a) Nechť $\alpha = e$. Potom podle lemmatu 5.2.9 existuje

$$C \to \alpha X \,.\, \delta \in I(\gamma X)$$

takové, že existuje pravé odvození

(*) $\delta \Rightarrow^* Bz$ pro nějaké $z \in \Sigma^*$.

Podle lemmatu 5.2.10 je

$$C \to \alpha \,.\, X\delta \in I(\gamma),$$

a tedy

$$C \to \alpha X \,.\, \delta \in M,$$

podle podmínky 1. Dále podle (*) existují

$$X_1, \ldots, X_k \in \Pi \text{ a } \xi, \tau_1, \ldots, \tau_k \in (\Pi \cup \Sigma)^*$$

takové, že

$$\delta = X_1\xi,\ X_1 \to X_2\tau_1 \in P,$$

$$X_2 \to X_3\tau_2 \in P, \ldots, X_k \to B\tau_k \in P.$$

Podle podmínky 2 jsou proto položky

$$X_1 \to .X_2\tau_1,$$

$$X_2 \to .X_3\tau_2, \ldots, X_k \to .B\tau_k,\ B \to .\beta \text{ prvky } M.$$

Dokázali jsme tedy, že $B \to .\beta \in M$.

b) Nechť $\alpha \neq e$. Potom podle lemmatu 5.2.10 je $\alpha = \alpha' X$, pro nějaké X a $B \to \alpha' \,.\, X\beta \in I(\gamma)$. Proto podle 1 je

$$B \to \alpha' X \,.\, \beta \in M.$$

Dokázali jsme, že $B \to \alpha \,.\, \beta \in M$.

Dohromady: $I(\gamma X) \subseteq M$.

$I(\gamma X)$ je tedy skutečně nejmenší množinou uvedených vlastností.

Věta 5.2.14. *Nechť $\mathscr{G} = (\Pi, \Sigma, S, P)$ je libovolná bezkontextová gramatika. Pro každé $\gamma \in (\Pi \cup \Sigma)^*$ opět označme $I(\gamma)$ množinu všech položek platných pro γ. Na $(\Pi \cup \Sigma)^*$ definujme ekvivalenci $\sim$ předpisem*

$$\gamma_1 \sim \gamma_2 \Leftrightarrow I(\gamma_1) = I(\gamma_2).$$

Potom $\sim$ je pravou kongruencí konečného indexu.

Důkaz. Relace $\sim$ je zřejmě ekvivalencí a je konečného indexu, neboť existuje jen konečně mnoho různých množin položek gramatiky $\mathscr{G}$. Podle lemmatu 5.2.13 je pro $I(\gamma_1) = I(\gamma_2)$ a libovolné $X \in \Pi \cup \Sigma$ také $I(\gamma_1 X) = I(\gamma_2 X)$. Jinak řečeno,

$$\gamma_1 \sim \gamma_2 \Rightarrow \gamma_1 X \sim \gamma_2 X \text{ pro libovolné } X,$$

tzn. $\sim$ je pravá kongruence.

Podle 1.3.3 a předchozí věty existuje ke každé bezkontextové gramatice $\mathscr{G}$ deterministický konečný automat takový, že dva řetězy převedou automat do téhož stavu, právě když jim odpovídají stejné množiny položek.

Konstrukce takového automatu je jednoduchá: nejprve se sestrojí počátečnímu stavu odpovídající množina $I(e)$ a množiny odpovídající dosažitelným stavům se postupně sestrojí podle lemmatu 5.2.13.

Zbývá charakterizovat množinu $I(e)$.

Lemma 5.2.15. *$I(e)$ je nejmenší množina M taková, že*

1. $M \supseteq \{S \to .\alpha;\ S \to \alpha \in P\}$,
2. *jestliže* $A \to .B\beta \in M$, *potom pro každé pravidlo* $B \to \vartheta \in P$ *je také* $B \to .\vartheta \in M$.

Důkaz. a) Podle definice položek platných pro e splňuje $I(e)$ podmínku 1 a podle lemmatu 5.2.8 splňuje i podmínku 2.

b) Budiž M libovolná množina splňující 1 i 2. Ukážeme, že $I(e) \subseteq M$. $I(e)$ zřejmě obsahuje pouze položky typu $X \to .\eta$, jak se ověří přímo z definice. Nechť tedy $X \to .\eta \in I(e)$. Potom existuje pravá větná forma Xu pro nějaké $u \in \Sigma^*$. To však znamená, že v P existují pravidla

$$S \to X_1\tau_1,\ X_1 \to X_2\tau_2, \ldots, X_k \to X\tau_k$$

pro nějaká

$$X_1, \ldots, X_k \in \Pi \text{ a } \tau_1, \ldots, \tau_k \in (\Pi \cup \Sigma)^*,\ k \geqq 0.$$

Podle podmínky 1 obsahuje M položku $S \to .X_1\tau_1$ a podle podmínky 2 zahrnuje i položky

$$X_1 \to .X_2\tau_2, \ldots, X_k \to .X\tau_k.$$

z posledně uvedené položky a z podmínky 2 vyplývá, že do M patří i $X \to .\eta$. Tedy $I(e) \subseteq M$, jak jsme měli dokázat.

Příklad 5.2.16. Sestrojme konečný automat (s přechodovou funkcí δ) charakterizující množiny $I(\gamma)$ pro gramatiku

$$S' \to A,$$
$$A \to [S],$$
$$A \to a,$$
$$S \to SA,$$
$$S \to A,$$

z př. 5.2.1. Jednotlivé stavy automatu budeme ztotožňovat s příslušnými množinami položek. Je tedy

$$q_0 = I(e): \; S' \to .A \quad \text{podle podmínky 1 lemmatu 5.2.15,}$$
$$\left.\begin{array}{l} A \to .[S] \\ A \to .a \end{array}\right\} \text{podle podmínky 2 lemmatu 5.2.15.}$$

Z e stavu q_0 jsou důležité pouze přechody na základě symbolů A, a, $[$.

$$q_1 = I(A): \; S' \to A.$$
$$q_2 = I(a): \; A \to a.$$
$$q_3 = I([): \; A \to [.S] \quad \text{podle podmínky 1 lemmatu 5.2.13,}$$
$$\left.\begin{array}{l} S \to .SA \\ S \to .A \\ A \to .[S] \\ A \to .a \end{array}\right\} \text{podle podmínky 2 lemmatu 5.2.13.}$$

Všechny ostatní přechody z q_0 vedou do „mrtvého" stavu q_m odpovídajícího prázdné množině položek, tzn.

$$I(S) = I([) = I(S') = q_m.$$

Pro tento stav podle 5.2.11 platí, že

$$\delta(q_m, X) = q_m \text{ pro každé } X \in \Pi \cup \Sigma.$$

Ze stavů q_1 a q_2 lze přejít pouze do stavu q_m. Ze stavu q_3 dávají jinou možnost jenom přechody na základě symbolů a, A, S, $[$. Podle lemmatu 5.2.13 dostáváme

$$\begin{aligned} q_4 &= I([A): S \to A.,\\ q_5 &= I([S): A \to [S.],\\ &\qquad S \to S.A,\\ &\qquad A \to .[S],\\ &\qquad A \to .a. \end{aligned}$$

Spočítáme-li $I([[)$, zjistíme, že nedostáváme nový stav, neboť $I([[) = q_3$, podobně $I([a) = q_2$. Ze stavu q_4 lze přejít opět jenom do q_m. Ze stavu q_5 se do jiného stavu než q_m přejde na základě symbolů $]$, A, a, $[$:

$$\begin{aligned} q_6 &= I([S]): A \to [S].,\\ q_7 &= I([SA): S \to SA.. \end{aligned}$$

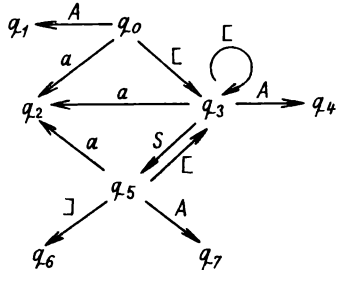
Zbývající dva přechody k novým stavům nevedou, neboť

$$I([Sa) = q_2 \text{ a } I([S[) = q_3 .$$

Protože ze stavů q_6 a q_7 je možné přejít jenom do stavu q_m, je konstrukce automatu u konce. Stavový diagram automatu je na obr. 42. Pro větší přehlednost je na diagramu vypuštěn stav q_m a všechny přechody do něho vedoucí.

Automat, jehož konstrukci jsme právě předvedli na příkladu, budeme nazývat položkovým automatem.

Definice 5.2.17. Nechť $\mathcal{G} = (\Pi, \Sigma, S, P)$ je libovolná bezkontextová gramatika. *Položkovým automatem pro gramatiku* $\mathcal{G}$ bu-



Obr. 42

demc nazývat konečný automat $\mathscr{A} = (Q, \Gamma, \delta, q_0, F)$ definovaný takto:

$$
\begin{aligned}
Q &= \{I(\gamma);\ \gamma \in (\Pi \cup \Sigma)^*\}\,,\\
\Gamma &= \Pi \cup \Sigma\,,\\
q_0 &= I(e)\,,\\
F &= \{I(\gamma);\ \gamma \in (\Pi \cup \Sigma)^* \ \&\ I(\gamma) \neq \emptyset\} = Q - \{\emptyset\}\,,\\
\delta(I(\gamma), a) &= I(\gamma a) \text{ pro všechna } a \in \Gamma.
\end{aligned}
$$

Víme už, že místo celého Q můžeme v položkovém automatu brát pouze stavy dosažitelné ze stavu q_0.

Z věty 5.2.14 plyne, že definice položkového automatu je korektní. Tento automat zřejmě přijímá právě ty řetězy terminálů a neterminálů, ke kterým existuje alespoň jedna platná položka. Pro naše další úvahy je důležitá i struktura stavů položkového automatu – každý stav je množinou položek. Z definice jeho přechodové funkce vyplývá, že pro každý řetěz $\gamma \in (\Pi \cup \Sigma)^*$ je $\delta(q_0, \gamma) = I(\gamma)$.

Poznámka 5.2.18. Položkový automat může sloužit jako zdroj dodatečné informace při analýze zdola. Jak už víme z čl. 4.6, při úspěšné analýze řetězu w dává v každém okamžiku obsah zásobníku η a dosud nepřečtený úsek u vstupního slova dohromady pravou větnou formu ηu vyskytující se při odvozování w, tj.

$$S \Rightarrow^* \eta u \Rightarrow^* w\,.$$

Navíc existuje pravidlo $A \to \alpha\beta$ a řetězy

$$\varepsilon \in (\Pi \cup \Sigma)^*,\ v \in \Sigma^*$$

tak, že

$$(*) \qquad S \Rightarrow_p^* \varepsilon A v \Rightarrow_p \varepsilon\alpha\beta v \text{ a } \eta = \varepsilon\alpha,\ \beta v \Rightarrow^* u\,.$$

Z (*) je vidět, že $A \to \alpha\,.\,\beta \in I(\eta)$. V následujícím kroku analýzy se

a) buď provede redukce, a to v případě, že $\beta = e$ (a $A \to \alpha.$ je úplná položka),

b) nebo přesune do zásobníku další vstupní symbol, a to v případě, že $\beta \neq e$ (a v položce $A \to \alpha\,.\,\beta$ je bezprostředně vpravo

od tečky terminální symbol souhlasící s terminálem, který se bude přesouvat do zásobníku).

Znalost množiny $I(\eta)$ v okamžiku, kdy v zásobníku je řetěz η, tedy může podstatně zúžit výběr dalších pokračování analýzy. V případě LR(0) gramatik, které nyní budeme definovat, už $I(\eta)$ dokonce poskytuje veškerou potřebnou dodatečnou informaci.

Definice 5.2.19. Bezkontextová gramatika

$$\mathcal{G} = (\Pi, \Sigma, S, P)$$

se nazývá *LR(0) gramatika*, jestliže splňuje tyto podmínky:

1. počáteční symbol S se nevyskytuje na pravé straně žádného pravidla;
2. pro libovolný řetěz $\gamma \in (\Pi \cup \Sigma)^*$ se v množině $I(\gamma)$ vyskytuje nejvýše jedna úplná položka;
3. jestliže se v $I(\gamma)$ vyskytuje úplná položka, potom už v $I(\gamma)$ není obsažena žádná položka, v níž je bezprostředně vpravo od tečky terminální symbol.

Poznámka 5.2.20. O každé bezkontextové gramatice je možné efektivně rozhodnout, zda je to LR(0) gramatika. Podmínka 1 se ověří přímo ze zadání pravidel gramatiky a podmínky 2 a 3 se zkontrolují podle položkového automatu, který lze ke každé gramatice sestrojit.

Označení LR(k) (zatím probíráme případ $k = 0$) opět pochází z anglického názvosloví (srov. poznámku 5.1.3) a napovídá, že se jedná o gramatiku umožňující analýzu, při níž je analyzované slovo čteno zleva doprava (left-to-right), je vydáván pravý rozbor (right parse) a analýza se opírá o prohlížení následujících k vstupních symbolů.

Příklad 5.2.21. Gramatika z př. 5.2.1 je LR(0). Podmínka 1 je splněna, a jak ukazuje př. 5.2.16, jsou splněny i podmínky 2 a 3. Úplné položky jsou zde totiž pouze ve stavech q_1, q_2, q_4, q_6 a q_7, které už žádné jiné položky neobsahují.

Poznámka 5.2.22. Poznámka 5.2.18 ukazuje, proč u LR(0) gramatik poskytuje položkový automat postačující dodatečnou informaci. Podle definice 5.2.19, podmínky 3, nemůže dojít k si-

tuací, že by položkový automat současně nabízel možnost redukovat zásobník a možnost přesunout do zásobníku další vstupní symbol. V případě, že automat určí jako následující krok redukci zásobníku, je podle 5.2.19, podmínky 2, jednoznačně určeno, podle kterého pravidla gramatiky se má redukce provádět. Konečně podmínka 1 zajišťuje, že v zásobníku se symbol S může vyskytnout pouze v situaci odpovídající konci výpočtu, tzn. zásobník obsahuje jediný symbol, a to S.

Při analýze zdola řetězu podle LR(0) gramatiky je — v situaci, kdy zásobník obsahuje řetěz η — jediné perspektivní pokračování určeno stavem $q = \delta(q_0, \eta)$ položkového automatu. Pokud q obsahuje úplnou položku, řekněme položku $A \to \alpha.$, potom se provádí redukce zásobníku — odebere se vrchních $|\alpha|$ symbolů (které ovšem tvoří α) a nahradí se symbolem A. Podle podmínky 2 definice LR(0) gramatiky je pravidlo redukce určeno jednoznačně. Jestliže stav q neobsahuje úplnou položku, provede se přesun vstupního symbolu do zásobníku.

Příklad 5.2.23. Do probíraných postupů rychleji vniknete, jestliže si znovu proberete výpočet z př. 5.2.1 a v každém kroku ověříte, že položkový automat z př. 5.2.16 skutečně poskytuje potřebnou dodatečnou informaci. Zde se omezíme pouze na vybrané dva kroky výpočtu.

1. Po skončení 7. kroku výpočtu obsahuje zásobník řetěz $\|SA$, který převádí položkový automat do stavu $\delta(q_0, [[SA) = q_7$. Tento stav obsahuje úplnou položku $S \to SA.$. V následujícím okamžiku se proto redukuje řetěz SA na vrcholu zásobníku na symbol S.

2. Po skončení 11. kroku výpočtu obsahuje zásobník řetěz $[S$, jenž převádí položkový automat do stavu $\delta(q_0, [S) = q_5$. Poněvadž tento stav neobsahuje úplnou položku, provede se přesun do zásobníku.

Místo aby se stav položkového automatu odpovídající obsahu η zásobníku počítal v každém okamžiku znovu, je možné modifikovat obhospodařování zásobníku tak, aby byl příslušný stav v každém okamžiku bezprostředně k dispozici. Při této modifikaci je místo řetězu $\eta = x_1 \dots x_n$ ($x_1, \dots, x_n$ označují jednotlivé

symboly) v zásobníku uložen řetěz $q_0x_1q_1x_2q_2 \dots x_nq_n$, kde q_0 je počáteční stav položkového automatu, a platí

(*) pro každé i, $1 \leqq i \leqq n$, je $q_i = \delta(q_0, x_1 \dots x_i)$.

Způsob, jakým se s takto obohaceným zásobníkem pracuje, ukazují následující pravidla.

1. Výpočet začíná se zásobníkem obsahujícím pouze symbol q_0.

2. Jestliže zásobník obsahuje řetěz $q_0x_1q_1x_2q_2 \dots x_nq_n$ splňující (*) a dalším krokem výpočtu je přesun dalšího vstupního symbolu y do zásobníku (tento další krok se určí na základě q_n), uloží se na vrchol zásobníku symbol y a q, kde $q = \delta(q_n, y)$, takže nový obsah zásobníku je $q_0x_1 \dots x_nq_nyq$ a tento řetěz opět splňuje vlastnost (*).

3. Jestliže zásobník obsahuje řetěz $q_0x_1q_1 \dots x_nq_n$ splňující (*) a dalším krokem výpočtu (určeným na základě q_n) je redukce podle pravidla $y \to x_i \dots x_n$ gramatiky (pro nějaké i, $1 \leqq i \leqq n$, a y různé od počátečního symbolu gramatiky), odstraní se z vrcholu zásobníku řetěz $x_iq_i \dots x_nq_n$ a nahradí se řetězem yq, kde $q = \delta(q_{i-1}, y)$, takže vznikne řetěz $q_0x_1q_1 \dots x_{i-1}q_{i-1}yq$, opět splňující (*).

Jestliže se provádí redukce podle pravidla typu $S \to \alpha$, kde S je počáteční symbol gramatiky, znamená to, že je třeba ukončit výpočet, neboť u LR(0) gramatik se počáteční symbol nevyskytuje na pravé straně žádného pravidla, takže redukce uvedeného typu signalizuje konec analýzy. Proto se v takovém okamžiku vyprázdní zásobník, který ovšem pod obohacenou verzí řetězu α obsahoval pouze symbol q_0.

Příklad 5.2.24. Popišme zásobníkový automat, přijímající jazyk generovaný gramatikou $\mathcal{G}$ z př. 5.2.1.

Vstupní abeceda:], [, a.

Zásobníková abeceda: $q_0, q_1, \dots, q_7$,], [, a, S', S, A.

Počáteční zásobníkový symbol: q_0.

Přechodová funkce a stavy automatu jsou implicitně obsaženy v popisu činnosti zadané tab. 14 (k tomu viz též následující poznámku 5.2.25). Funkce δ, na níž se tabulka odvolává, je přechodová funkce položkového automatu z př. 5.2.16 a obr. 42.

ıslo ıce	Symbol na vrcholu zásobníku	Prováděná akce	Podrobnější komentář	Výstupní tisk (číslo pravidla)
1	q_0	přesun do zásobníku	do zásobníku se vloží další vstupní symbol y, nad něj se uloží $q' = \delta(q_0, y)$	–
2	q_1	redukce podle pravidla $S' \to A$	konec výpočtu – ze zásobníku se odstraní 3 symboly (nutně: $q_0 A q_1$), čímž se vyprázdní zásobník	1
3	q_2	redukce podle pravidla $A \to a$	ze zásobníku se odstraní 2 symboly, tím se objeví jisté q, spočítá se $q' = \delta(q, A)$ a do zásobníku se uloží Aq'	3
4	q_3	přesun do zásobníku	do zásobníku se vloží další vstupní symbol y, nad něj se uloží $q' = \delta(q_3, y)$	–
5	q_4	redukce podle pravidla $S \to A$	ze zásobníku se odstraní 2 symboly, tím se objeví jisté q, nad něj se uloží Sq', kde $q' = \delta(q, S)$	5

Tab. 14

Číslo akce	Symbol na vrcholu zásobníku	Prováděná akce	Podrobnější komentář	Výstupní tisk (číslo pravidla)
6	q_5	přesun do zásobníku	do zásobníku se vloží yq', kde y je další vstupní symbol a $q' = \delta(q_5, y)$	–
7	q_6	redukce podle pravidla $A \to [S]$	ze zásobníku se odstraní 6 symbolů, tím se objeví jisté q, nad něj se uloží Aq', kde $q' = \delta(q, A)$	2
8	q_7	redukce podle pravidla $S \to SA$	ze zásobníku se odstraní 4 symboly, tím se objeví jisté q, nad něj se uloží Sq', kde $q' = \delta(q, S)$	4

Poznámka 5.2.25. Analýza LR(0) gramatik, kterou jsme právě popsali, užívá i operací, které nemohou být provedeny jedinou instrukcí deterministického zásobníkového automatu, jak jsme ho definovali v 4.4.7. Každou takovou operaci však lze simulovat několika kroky vhodného deterministického automatu. Ilustrujme si to na příkladu analyzátoru z př. 5.2.24. Příslušný automat by měl stav p, v němž by zahajoval každou akci z tab. 14 a do něhož by se po skončení akce vracel. Pro každou akci by měl vyhraženo několik dalších stavů. Tak např. akci č. 8 by realizoval pomocí této soustavy instrukcí:

$(p, e, q_7) \to (p_8^3, e)$,

$(p_N^A, e, X) \to (p_8^2, e)$ pro každé $X \in \{], [, a, S, A\}$,
$(p_N^A, e, X) \to (p_8^1, e)$ pro každé $X \in \{q_0, \ldots, q_7\}$,
$(p_N^A, e, X) \to (p_8^S, e)$ pro každé $X \in \{], [, a, S, A\}$,
} odebírání čtyř symbolů

$(p_N^N, e, q) \to (p, Sq')$ pro každé $q \in \{q_0, \ldots, q_7\}$ a $q' = \delta(q, S)$, pokud je $\delta(q, S)$ definováno.

Stavy p_8^3, p_8^2, p_8^1, p_8^S se v jiných instrukcích nebudou vyskytovat. Podobně např. akce č. 4 by se realizovala touto soustavou instrukcí:

$$(p, [, q_3) \to (p, q_3[q_3),$$
$$(p, a, q_3) \to (p, q_3 a q_2).$$

$\delta(q_3,])$ není definována, a proto příslušná instrukce chybí.

Z těchto úvah přímo vyplývá následující tvrzení.

Věta 5.2.26. *Ke každé* LR(0) *gramatice* $\mathscr{G}$ *lze sestrojit deterministický zásobníkový automat* $\mathscr{M}$ *takový, že* $L(\mathscr{G}) = N(\mathscr{M})$.

Poznámka 5.2.27. Příslušný deterministický zásobníkový automat je možné rozšířit o jednocestnou výstupní pásku, na níž se v okamžiku, kdy provádí redukci podle i-tého pravidla gramatiky, vytiskne číslo i a hlava se posune doprava. Jestliže takový automat přijme vstupní slovo w, bude na konci výpočtu obsahovat výstupní páska pravý rozbor slova w.

Příklad 5.2.28. 1. Zásobníkový automat zadaný v př. 5.2.24 provede nad vstupním slovem $[[aa]\, a]$ výpočet charakterizovaný touto posloupností situací (viz tab. 14 v př. 5.2.24 a obr. 42):

$([[aa]\, a], q_0, -)$,
$([aa]\, a], q_0[q_3, -)$,
$(aa]\, a], q_0[q_3[q_3, -)$,
$(a]\, a], q_0[q_3[q_3 a q_2, -)$,
$(a]\, a], q_0[q_3[q_3 A q_4, 3)$,
$(a]\, a], q_0[q_3[q_3 S q_5, 35)$,
$(]\, a], q_0[q_3[q_3 S q_5 a q_2, 35)$,

$([]\,a],\ q_0[q_3[q_3Sq_5Aq_7,\ 353),$
$([]\,a],\ q_0[q_3[q_3Sq_5,\ 3534),$
$(a],\ q_0[q_3[q_3Sq_5]\,q_6,\ 3534),$
$(a],\ q_0[q_3Aq_4,\ 35342),$
$(a],\ q_0[q_3Sq_5,\ 353425),$
$([],\ q_0[q_3Sq_5aq_2,\ 353425),$
$([],\ q_0[q_3Sq_5Aq_7,\ 3534253),$
$([],\ q_0[q_3Sq_5,\ 35342534),$
$(-,\ q_0[q_3Sq_5]\,q_6,\ 35342534),$
$(-,\ q_0Aq_1,\ 353425342),$
$(-,\ -,\ 3534253421).$

Výsledkem výpočtu je přijetí slova a vytištění jeho pravého rozboru 3534253421.

2. Nad slovem $[]\,aa]\,a]$ provede tento automat výpočet charakterizovaný posloupností situací

$([]\,aa]\,a],\ q_0,\ -),$
$([]\,aa]\,a,\ q_0[q_3,\ -).$

Tam výpočet končí – neboť $\delta(q_3,])$ není definováno –, což znamená, že vstupní řetěz nepatří do analyzovaného jazyka (obsahuje syntaktickou chybu).

ROZŠIŘUJÍCÍ ČÁST

5.3. Vztah LR(0) gramatik a deterministických jazyků

V čl. 5.2 jsme ukázali, že ke každé LR(0) gramatice $\mathscr{G}$ existuje deterministický zásobníkový automat $\mathscr{M}$ takový, že $N(\mathscr{M}) = L(\mathscr{G})$. V tomto článku dokážeme, že platí i obrácené tvrzení.

Věta 5.3.1. *Ke každému deterministickému zásobníkovému automatu $\mathscr{M}$ lze sestrojit* LR(0) *gramatiku $\mathscr{G}$ takovou, že $L(\mathscr{G}) = N(\mathscr{M})$.*

Před vlastním důkazem věty uvedeme a dokážeme několik pomocných tvrzení.

Nejprve popíšeme postup, kterým se příslušná gramatika získá.

Konstrukce 5.3.2. Budiž $\mathscr{M} = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ deterministický zásobníkový automat. K tomuto automatu sestrojíme gramatiku postupem popsaným ve větách 4.5.3 a 4.5.5. To znamená, že se jedná o gramatiku $\mathscr{G}'$ danou systémem pravidel:

$$S \to \langle q_0, Z_0, q \rangle$$

pro všechna $q \in Q$,

$$\langle q, X, q' \rangle \to a \langle q_1, X_1, q_2 \rangle \langle q_2 X_2 q_3 \rangle \dots \langle q_k, X_k, q' \rangle$$

pro všechna $k \geqq 0$,

$$a \in \Sigma \cup \{e\}, \quad q, q_1, \dots, q_k, q' \in Q$$

taková, že

$$\delta(q, a, X) = (q_1, X_1 \dots X_k).$$

Tuto gramatiku zredukujeme podle lemmatu 4.1.2, čímž získáme výslednou gramatiku $\mathscr{G}$. Pro ni je $L(\mathscr{G}) = N(\mathscr{M})$.

Příklad 5.3.3. Mějme deterministický zásobníkový automat $\mathscr{M}$ s počátečním zásobníkovým symbolem A a přechodovou funkcí

$$\delta(q_0, a, A) = (q_0, AB),$$
$$\delta(q_0, b, A) = (q_0, BA),$$
$$\delta(q_0, e, B) = (q_1, AB),$$
$$\delta(q_1, a, B) = (q_1, e),$$
$$\delta(q_1, e, A) = (q_1, e).$$

Podle 4.5.5 sestrojíme gramatiku $\mathscr{G}'$:

$$S \to \langle q_0, A, q_0 \rangle \mid \langle q_0, A, q_1 \rangle,$$
$$\langle q_0, A, q_0 \rangle \to a \langle q_0, A, q_0 \rangle \langle q_0, B, q_0 \rangle \mid$$
$$\mid a \langle q_0, A, q_1 \rangle \langle q_1, B, q_0 \rangle \mid$$
$$\mid b \langle q_0, B, q_0 \rangle \langle q_0, A, q_0 \rangle \mid$$
$$\mid b \langle q_0, B, q_1 \rangle \langle q_1, A, q_0 \rangle,$$

$$
\begin{aligned}
\langle q_0, A, q_1\rangle &\to a\langle q_0, A, q_0\rangle\langle q_0, B, q_1\rangle \mid \\
&\quad \mid a\langle q_0, A, q_1\rangle\langle q_1, B, q_1\rangle \mid \\
&\quad \mid b\langle q_0, B, q_0\rangle\langle q_0, A, q_1\rangle \mid \\
&\quad \mid b\langle q_0, B, q_1\rangle\langle q_1, A, q_1\rangle , \\
\langle q_0, B, q_0\rangle &\to \langle q_1, A, q_0\rangle\langle q_0, B, q_0\rangle \mid \\
&\quad \mid \langle q_1, A, q_1\rangle\langle q_1, B, q_0\rangle , \\
\langle q_0, B, q_1\rangle &\to \langle q_1, A, q_0\rangle\langle q_0, B, q_1\rangle \mid \\
&\quad \mid \langle q_1, A, q_1\rangle\langle q_1, B, q_1\rangle , \\
\langle q_1, B, q_1\rangle &\to a , \\
\langle q_1, A, q_1\rangle &\to e .
\end{aligned}
$$

Po redukci podle 4.1.2 získáme gramatiku $\mathscr{G}$:

$$
\begin{aligned}
S &\to \langle q_0, A, q_1\rangle , \\
\langle q_0, A, q_1\rangle &\to a\langle q_0, A, q_1\rangle\langle q_1, B, q_1\rangle \mid \\
&\quad \mid b\langle q_0, B, q_1\rangle\langle q_1, A, q_1\rangle , \\
\langle q_0, B, q_1\rangle &\to \langle q_1, A, q_1\rangle\langle q_1, B, q_1\rangle , \\
\langle q_1, B, q_1\rangle &\to a , \\
\langle q_1, A, q_1\rangle &\to e .
\end{aligned}
$$

K tomu, abychom dokázali větu 5.3.1, stačí ověřit, že gramatika získaná podle konstrukce 5.3.2 je vždy LR(0). Při tom budeme používat značení zavedeného v konstrukci 5.3.2.

Použijeme vlastností, které má gramatika $\mathscr{G}$ díky tomu, že byla sestrojena na základě deterministického automatu.

Pro každý symbol x gramatiky $\mathscr{G}$ definujeme *charakteristiku* $[x]$ *symbolu* x takto:

$$
\begin{aligned}
[x] &= \Sigma , && \text{jestliže } x \text{ je terminál, tj. } x \in \Sigma ; \\
[x] &= (q, Y) , && \text{jestliže } x = \langle q, Y, q'\rangle \text{ pro nějaké } q' ; \\
[x] &= S , && \text{jestliže } x = S .
\end{aligned}
$$

Charakteristiku řetězů α utvořených z terminálů i neterminálů gramatiky $\mathscr{G}$ definujeme předpisem

$$
\begin{aligned}
[\alpha] &= e , && \text{jestliže } \alpha = e , \\
[\alpha] &= [x] , && \text{jestliže } \alpha = x\alpha' \text{ pro nějaké } \alpha' .
\end{aligned}
$$

Charakteristiku položky $p = X \to \alpha . \beta$ definujeme takto:

$$[p] = ([X], \alpha, [\beta]) .$$

Z determinismu výchozího automatu $\mathscr{M}$ vyplývá důležitá vlastnost gramatiky $\mathscr{G}$:

Lemma 5.3.4. *Jestliže* (c_1^1, α^1, c_2^1), (c_1^2, α^2, c_2^2) *jsou charakteristiky dvou položek gramatiky* $\mathscr{G}$, *potom*

$$c_1^1 = c_1^2 \,\&\, \alpha^1 = \alpha^2 \Rightarrow c_2^1 = c_2^2 ,$$

tzn. první dvě složky charakteristiky libovolné položky jednoznačně určují složku třetí.

Důkaz. Budiž $p = X \to \alpha . \beta$ položka gramatiky $\mathscr{G}$ a $[X] = -(q, Y)$ pro nějaké $q \in Q$ a $Y \in \Gamma$. Potom pravidlo $X \to \alpha\beta$ odpovídá instrukci $\delta(q, a, Y) = (q', Y_1 \dots Y_k)$ automatu $\mathscr{M}$, pro nějaká $a \in \Sigma \cup \{e\}$ a $Y_1 \dots Y_k \in \Gamma^*$. Jelikož $\mathscr{M}$ je deterministický, určuje q a Y, zda $a = e$, nebo $a \in \Sigma$.

a) Jestliže $a = e$, jsou už z q a Y určeny i q' a $Y_1 \dots Y_k$. Pro $k = 0$ je pochopitelně $[\beta] = e$. Nechť tedy $k \geqq 1$.

Jestliže $\alpha = e$, je tím určeno $[\beta] = (q', Y_1)$. Pokud

$$\alpha = \langle q', Y_1, q_1 \rangle \langle q_1, Y_2, q_2 \rangle \dots \langle q_{i-1}, Y_i, q_i \rangle$$

pro nějaké $i \geqq 1$ a $q_1, \dots, q_i \in Q$, je

$$[\beta] = (q_i, Y_{i+1}), \text{ jestliže } i < k ,$$

a $|\beta| \quad e$, pokud $i = k$.

b) V případě, kdy (q, Y) určí, že $a \in \Sigma$, mohou nastat dvě možnosti:

$\alpha \quad e$: v tom případě je $[\beta] = \Sigma$;

$\alpha \neq e$: v tomto případě první symbol a řetězu spolu s (q, Y) určuje jak q', tak $Y_1 \dots Y_k$.

Pokud $\alpha = a$, je $[\beta] = (q', Y_1)$, nebo $[\beta] = e$, jestliže

$$Y_1 \dots Y_k = e .$$

Pokud

$$\alpha = a \langle q', Y_1, q_1 \rangle \dots \langle q_{i-1}, Y_i, q_i \rangle$$

pro nějaké i, $k > i \geqq 1$ (resp. $k = i$), je

$$[\beta] = (q_i, Y_{i+1})$$

(resp. $[\beta] = e$). V každém případě tedy lze na základě δ určit z prvních dvou složek charakteristiky $[p]$ i její třetí složku.

Pro zvláštní případ položek typu

$$S \to .\langle q_0, Z_0, q\rangle \text{ a } S \to \langle q_0, Z_0, q\rangle .$$

lemma očividně také platí.

Příklad 5.3.5. U gramatiky z př. 5.3.3 je

$$R = \{\Sigma, S, (q_0, A), (q_1, A), (q_0, B), (q_1, B)\}$$

množinou charakteristik symbolů gramatiky a $R \cup \{e\}$ množinou charakteristik řetězů. Charakteristikou položky

$$\langle q_0, A, q_1\rangle \to a\langle q_0, A, q_1\rangle . \langle q_1, B, q_1\rangle$$

je trojice

$$((q_0, A), a\langle q_0, A, q_1\rangle, (q_1, B))$$

atd. Lemma 5.3.4 ilustrujme na příkladu neúplných charakteristik

$$((q_0, A), e, \bullet),\ ((q_0, A), b\langle q_0, B, q_1\rangle, \bullet) \text{ a } ((q_1, B), a, \bullet) .$$

Jejich třetími složkami jsou po řadě Σ, (q_1, A) a e.

Lemmatu 5.3.4 využijeme ke zkoumání struktury stavů $I(\gamma)$ položkového automatu pro gramatiku $\mathscr{G}$ (definovaného v 5.2.17).

Charakteristikou množiny položek M gramatiky $\mathscr{G}$ budeme nazývat

$$[M] = \{[p];\ p \in M\} .$$

Řekneme, že M *má souvislou charakteristiku*, jestliže

$$M = \{(c, \alpha, c_1), (c_1, e, c_2), (c_2, e, c_3), \ldots, (c_k, e, c_{k+1})\}$$

pro nějaká $c, c_1, \ldots, c_{k+1}$ a nějaké α.

Lemma 5.3.6. *Nechť* M *má souvislou charakteristiku a nechť*

$$p = A \to \alpha . B\beta \in M .$$

Potom má souvislou charakteristiku také množina

$$M \cup M_p,$$

kde

$$M_p = \{B \to .\vartheta;\ B \to \vartheta \text{ je pravidlo gramatiky } \mathscr{G}\}.$$

Důkaz. Všechny položky z M_p mají charakteristiku $([B], e, c)$, kde c je jednoznačně určeno z $[B]$, tzn.

$$[M_p] = \{([B], e, c)\}.$$

Dále je $([A], \alpha, [B]) \in M$, neboť $A \to \alpha . B\beta \in M$. Je tedy $[B]$ zároveň třetí složkou jisté charakteristiky z $[M]$ a první složkou charakteristiky z $[M_p]$. Z toho už plyne, že $M \cup M_p$ má souvislou charakteristiku.

Důsledek 5.3.7. *Počáteční stav $I(e)$ položkového automatu pro gramatiku $\mathscr{G}$ má souvislou charakteristiku.*

Důkaz. Množina $\{S \to .\langle q_0, Z_0, q\rangle;\ q \in Q\}$ má souvislou charakteristiku $\{(S, e, (q_0, Z_0))\}$. Tvrzení proto okamžitě vyplývá z lemmatu 5.2.15 a 5.3.6.

Lemma 5.3.8. *Nechť M má souvislou charakteristiku. Potom pro každý symbol x gramatiky $\mathscr{G}$ má také*

$$M_x = \{A \to \alpha x . \beta;\ A \to \alpha . x\beta \in M\}$$

souvislou charakteristiku.

Důkaz. Ukažme nejprve, že v $[M]$ je jediná trojice, jejíž třetí složka je $[x]$. Kdyby totiž existovaly dvě takové trojice, pak by v M existovaly charakteristiky typu

$$(\bullet, \bullet, [x]), ([x], e, [X_1]),$$
$$([X_1], e, [X_2]), \ldots, ([X_k], e, [x])$$

pro nějaké neterminály $X_1, \ldots, X_k$ gramatiky $\mathscr{G}$. Při determinismu M to znamená, že pro každý neterminál Y některé z charakteristik $[x], [X_1], \ldots, [X_k]$ obsahuje $\mathscr{G}$ pouze pravidla tvaru $Y \to X\tau$, kde X má opět jednu z těchto charakteristik. Proto Y nelze použít v žádné derivaci terminálního řetězu, což je spor s před-

pokladem, že gramatika $\mathscr{G}$ je redukovaná. M proto obsahuje jedinou trojici $([A], \alpha, [x])$, jejíž třetí složka je $[x]$. Proto $[M_x] = \{([A], \alpha x, c)\}$, kde c je jednoznačně určeno podle lemmatu 5.3.4. $[M_x]$ je jednobodová, a proto má souvislou charakteristiku.

Důsledek 5.3.9. *Každý stav $I(\gamma)$ položkového automatu pro gramatiku $\mathscr{G}$ má souvislou charakteristiku.*

Důkaz. Vyplývá okamžitě z lemmat 5.3.7, 5.2.13, 5.3.6 a 5.3.8. Z předchozího tvrzení už se snadno dokáže, že $\mathscr{G}$ je LR(0) gramatika.

Důkaz věty 5.3.1. Z definice množiny souvislé charakteristiky vyplývá, že taková množina může obsahovat nejvýše jednu trojici, jejíž třetí složka je e nebo Σ. Každá úplná položka $A \to \alpha.$ má charakteristiku $([A], \alpha, e)$. Každá položka $A \to \alpha.\beta$, v níž je napravo od tečky terminál, má charakteristiku $([A], \alpha, \Sigma)$. Proto každá množina souvislé charakteristiky může zahrnovat nejvýše jednu úplnou položku a v tom případě žádnou položku s terminálem bezprostředně napravo od tečky.

Každá množina $I(\gamma)$ je podle tvrzení 5.3.9 souvislé charakteristiky, a proto splňuje podmínku 2 definice LR(0) gramatiky (viz 5.2.19). Podmínka 1 je splněna také, neboť počáteční neterminál S se v žádném pravidle nevyskytuje na pravé straně.

Z vět 5.2.26 a 5.3.1 dostáváme následující výsledek.

Věta 5.3.10. *Jazyk lze popsat* LR(0) *gramatikou, právě když jej lze rozpoznat deterministickým zásobníkovým automatem prázdným zásobníkem.*

5.4. LR(k) gramatiky

V čl. 5.3 jsme zavedli položkový automat, jakožto zdroj informace použitelné k deterministické analýze LR(0) gramatik. Stavy tohoto automatu jsou množiny položek.

Nyní zavedeme obecnější pojem k-položky, který umožní příslušnou informaci získávat pro analýzu širší třídy gramatik.

Definice 5.4.1. Nechť $\mathscr{G} = (\Pi, \Sigma, S, P)$ je bezkontextová gramatika a $k \geq 1$ přirozené číslo. *k-položkou* gramatiky $\mathscr{G}$ nazveme

každou dvojici $(A \to \alpha.\beta, L)$, kde $A \to \alpha.\beta$ je položka $\mathscr{G}$ a $L \subseteq \Sigma^{*k}$. Symbolem Σ^{*k} označujeme množinu všech slov v abecedě Σ, jejichž délka nepřesahuje k. V dalším textu už budeme odkaz na gramatiku $\mathscr{G}$ vynechávat.

Definice 5.4.2. Řekneme, že k-položka $(A \to \alpha.\beta, L)$ je *platná k-položka* pro řetěz $\omega \in (\Sigma \cup \Pi)^*$, jestliže existuje pravá větná forma ηAu taková, že $\eta\alpha = \omega$ a $k(u) \in L$.

Poznámka 5.4.3. Jestliže $(A \to \alpha.\beta, L)$ je platná k-položka pro řetěz ω, potom je zřejmě $A \to \alpha.\beta$ platná položka pro ω.

Úmluva 5.4.4. Symbolem $I_k(\gamma)$ budeme označovat množinu všech platných k-položek pro řetěz γ.

Definice 5.4.5. Nechť $\mathscr{K}'$ je množina všech množin k-položek (pro danou gramatiku). Definujme funkci δ': $\mathscr{K}' \times (\Sigma \cup \Pi) \to \mathscr{K}'$ takto: pro každé $K \in \mathscr{K}'$ a $x \in (\Sigma \cup \Pi)$ je $\delta'(\mathscr{K}, x)$ nejmenší množina M taková, že

1. $M \supseteq \{(Y \to \alpha x.\beta, L); (Y \to \alpha.x\beta, L) \in K\}$;
2. jestliže $(A \to \alpha.B\beta, L) \in M$, potom pro každé pravidlo $B \to \vartheta$ gramatiky je $(B \to .\vartheta, k(\beta L)) \in M$.

Definice 5.4.6. Definujme množinu $K_0 \in \mathscr{K}'$, kde $\mathscr{K}'$ má týž význam jako v předchozí definici, jakožto nejmenší množinu M takovou, že

1. $M \supseteq \{(S \to .\alpha, \{e\}); S \to \alpha \in P\}$;
2. jestliže $(A \to .B\beta, L) \in M$, potom také pro každé pravidlo $B \to \vartheta \in P$ je $(B \to .\vartheta, k(\beta L)) \in M$.

Symbolem $\mathscr{K}$ označme množinu prvků množiny $\mathscr{K}'$ dosažitelných z K_0 (přechodovou funkcí δ') a symbolem δ označme restrikci δ' na $\mathscr{K} \times (\Sigma \cup \Pi)$.

Konečný automat

$$\mathscr{A} = (\mathscr{K}, \Sigma \cup \Pi, \delta, K_0, F),$$

kde $F = \mathscr{K} - \{\emptyset\}$, nazveme *k-položkovým automatem pro danou gramatiku.*

Definice 5.4.7. Nechť $k > 0$. Bezkontextová gramatika

$$\mathcal{G} = (\Pi, \Sigma, S, P)$$

je LR(k) *gramatika*, jestliže platí tyto tři podmínky:

1. Počáteční symbol se nevyskytuje na pravé straně žádného pravidla;

2. jestliže $(A \to \alpha., L_1)$, $(B \to \beta., L_2)$ jsou dvě k-položky (úplné k-položky) vyskytující se současně v nějakém stavu $K \in \mathcal{K}$ a takové, že $L_1 \cap L_2 \neq \emptyset$, potom pravidlo $A \to \alpha$ je totožné s pravidlem $B \to \beta$;

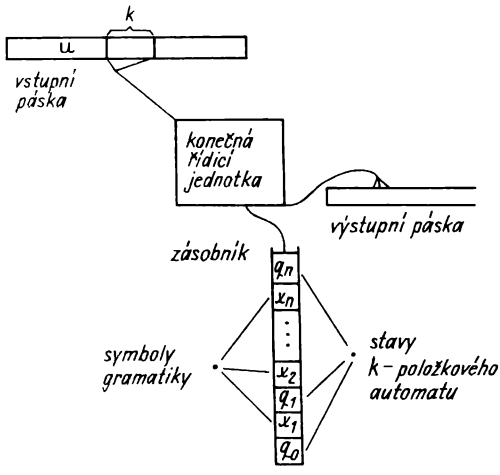
3. jestliže se v některém stavu $K \in \mathcal{K}$ současně s úplnou k-položkou $(A \to \alpha., L_1)$ vyskytne k-položka tvaru $(B \to \beta.\gamma, L_2)$, v níž se bezprostředně napravo od tečky vyskytuje terminál, potom $L_1 \cap k(\gamma L_2) = \emptyset$.

Poznámka 5.4.8. Z definic 5.2.17 a 5.4.6 vyplývá, že pro každý řetěz γ je

$$\{A \to \alpha.\beta;\ (A \to \alpha.\beta, L) \in \delta(K_0, \gamma) \text{ pro nějaké } L\} \subseteq I(\gamma).$$

Z tohoto zjištění a z definice 5.4.7 pak plyne, že každá LR(0) gramatika je LR(k) gramatikou pro libovolné $k \geqq 1$.

Analogicky lze dokázat, že každá LR(k) gramatika je LR($k+1$) pro každé $k \geqq 0$.



Obr. 43

Jazyk generovaný libovolnou LR(k) gramatikou

$$\mathcal{G} = (\Pi, \Sigma, S, P)$$

je možné analyzovat typem deterministického analyzátoru, který je schematicky zachycen na obr. 43.

Takový analyzátor je modifikací deterministického automatu s výstupní páskou. Změna se týká pouze vstupní hlavy, která v tomto případě snímá zároveň obsah k sousedních políček. Po zpracování libovolného počátečního úseku vstupního slova má proto LR(k) *analyzátor* k dispozici informaci i o následující k-tici vstupních symbolů. Zásobník je obsluhován obdobným způsobem, s jakým jsme se setkali už v čl. 5.3 u deterministické analýzy LR(0) jazyků. Díky tomu je v každém okamžiku výpočtu obsah zásobníku tvaru

$$q_0 x_1 q_1 x_2 q_2 \cdots q_{n-1} x_n q_n$$

(pravý konec opět odpovídá vrcholu zásobníku), kde $q_0, \ldots, q_n$ jsou stavy k-položkového automatu pro $\mathcal{G}$, $x_1, \ldots, x_n \in \Sigma \cup \Pi$, $n \geq 0$, q_0 je počáteční stav a

$$q_i = \delta(q_0, x_1 \ldots x_i)$$

pro všechna i, $1 \leqq i \leqq n$.

Způsob, jakým se obsah zásobníku aktualizuje při přenosu dalšího vstupního symbolu do zásobníku i při redukci obsahu podle některého pravidla gramatiky zůstává naprosto stejný jako u LR(0) analýzy.

Změna nastává pouze u postupu, kterým se v každém kroku výpočtu rozhoduje, zda dojde k přesunu do zásobníku nebo k redukci a ve druhém případě pak ještě podle kterého pravidla se redukce provádí.

U LR(0) gramatik se příslušné rozhodování provádělo na základě položky nacházející se na vrcholu zásobníku. U LR(k) analyzátoru se další postup určí podle těchto dvou údajů:

1. množiny q k-položek nacházející se na vrcholu zásobníku;
2. následující k-tice u vstupních symbolů.

Z nich se příští krok určí takto:

a) Jestliže q obsahuje k-položku $(A \to \alpha., L)$ takovou, že $u \in L$

a A není počáteční symbol, provede se redukce podle pravidla $A \to \alpha$. [Podrobněji: ze zásobníku se odebere vrchních 2 . $|\alpha|$ symbolů, tj. symboly tvořící α a stavy umístěné nad každým z nich. Tím se na vrcholu zásobníku objeví jistý stav q'. Nad něj se uloží dvojice symbolů Aq'', kde $q'' = \delta(q', A)$. Na výstupu se vytiskne číslo pravidla $A \to \alpha$.]

Jestliže A je počáteční symbol, ukončí se po provedení redukce výpočet vyprázdněním zásobníku a přijetím slova.

b) Jestliže q obsahuje k-položku $(A \to \alpha . \beta, L)$ takovou, že prvním symbolem β je terminál a $u \in k(\beta L)$, potom se provede přesun do zásobníku. [Podrobněji: do zásobníku se přesune další vstupní symbol b – který je nutně totožný s prvním symbolem β – a nad něj se uloží symbol $\delta(q, b)$.]

c) Jestliže nenastane ani možnost a), ani b), ukončí se výpočet zamítnutím zkoumaného slova.

Vlastnosti LR(k) gramatiky požadované v definici 5.4.7 zaručují, že ze všech eventualit uvedených sub a) až c) nastane právě jedna.

Příklad 5.4.9. Přesvědčíme se, že gramatika $\mathscr{G}$:

1. $S \to AB$,
2. $S \to A$,
3. $A \to aAb$,
4. $A \to e$,
5. $B \to bB$,
6. $B \to c$

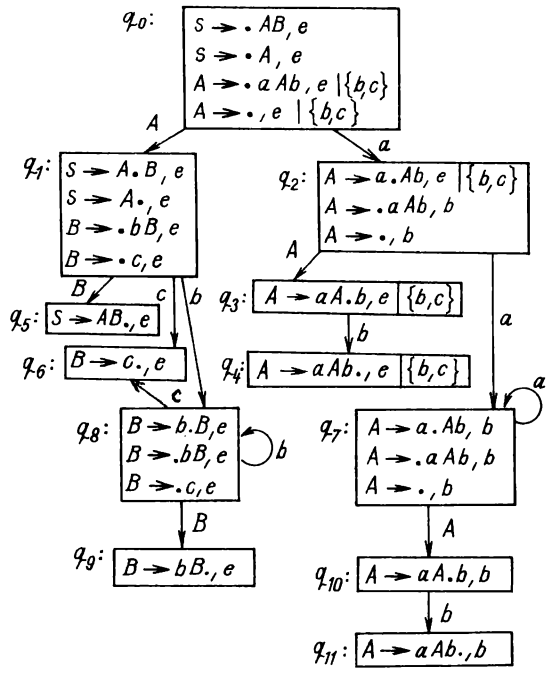
je LR(1) gramatika a sestrojíme k ní LR(1) analyzátor.

Diagram 1-položkového automatu pro $\mathscr{G}$ je sestrojen na obr. 44. Na diagramu jsme opět pro přehlednost vypustili stav odpovídající prázdné množině 1-položek (zde by to byl stav q_{12}) a všechny přechody do něj vedoucí. V diagramu je použito vžitého zkratkového značení. Jednoprvkový jazyk $\{x\}$ značíme pouze x; množinu položek

$$(A \to \alpha . \beta, L_1), \ldots, (A \to \alpha . \beta, L_n)$$

zkráceně zapisujeme

$$A \to \alpha . \beta, L_1 \mid L_2 \mid \ldots \mid L_n .$$



Obr. 44

Gramatika $\mathcal{G}$ je LR(1), neboť počáteční symbol se nevyskytuje na pravé straně žádného pravidla a 1-položkový automat na obr. 44 také splňuje podmínky z definice 5.4.7. Tak např. ve stavu q_0 jsou obsaženy dvě úplné položky

$$p_1 = (A \to ., e) \text{ a } p_2 = (A \to ., \{b, c\})$$

a dvě položky s terminálem bezprostředně napravo od tečky:

$$p_3 = (A \to .aAb, e) \text{ a } p_4 = (A \to .aAb, \{b, c\}) .$$

Ke konfliktu mezi p_1 a p_2 (viz podmínku 2 z definice 5.4.7) nedochází, neboť odpovídají témuž pravidlu a navíc

$$\{e\} \cap \{b, c\} = \emptyset .$$

Podobně nedochází ke konfliktu mezi p_1 a p_3 nebo p_4 (podmínka 3), protože

$$\{e\} \cap 1(aAb\{e\}) = \{e\} \cap 1(aAb\{b, c\}) = \emptyset .$$

Ke konfliktu mezi p_2 a p_3 nebo p_4 nedochází, protože

$$\{b, c\} \cap 1(aAb\{e\}) = \{b, c\} \cap 1(aAb\{b, c\}) = \emptyset .$$

Na základě 1-položkového automatu z obr. 44 lze tedy sestrojit LR(1) analyzátor pro gramatiku $\mathcal{G}$. Akce tohoto analyzátoru v závislosti na vrchním symbolu v zásobníku (tj. stavu 1-položkového

	Vstupní symbol			
	a	*b*	*c*	*e*
q_0	**přesun** *a*	$A \to e$; 4	$A \to e$; 4	$A \to e$; 4
q_1		**přesun** *b*	**přesun** *c*	**přijmout**; 2
q_2	**přesun** *a*	$A \to e$; 4		
q_3		**přesun** *b*		
q_4		$A \to aAb$; 3	$A \to aAb$; 3	$A \to aAb$; 3
q_5				**přijmout**; 1
q_6				$B \to c$; 6
q_7	**přesun** *a*	$A \to e$; 4		
q_8		**přesun** *b*	**přesun** *c*	
q_9				$B \to bB$; 5
q_{10}		**přesun** *b*		
q_{11}		$A \to aAb$; 3		
q_{12}				

Tab. 15

automatu) a následujícím vstupním symbolu (nebo e, což odpovídá tomu, že slovo bylo dočteno do konce) popíšeme tab. 15.

Akce označovaná v tabulce „$X \to \eta$; i" znamená: zredukovat podle pravidla $X \to \eta$ (včetně příslušné aktualizace zásobníku) a vytisknout i. Vstupní hlava se neposouvá.

Akce označovaná „**přijmout**; i" znamená: vyprázdnit zbytek zásobníku a vytisknout i. Výpočet končí přijetím slova.

Akce označovaná „**přesun** x" znamená: uložit symbol x do zásobníku (a přidat nad něj příslušný symbol stavu položkového automatu) a posunout vstupní hlavu.

Prázdná políčka v tabulce označují situace, ve kterých analyzátor končí a vydá hlášení, že analyzované slovo nepatří do $L(\mathcal{G})$. Speciálně taková situace nastává vždy, když se na zásobníku objeví symbol q_{12} odpovídající prázdné množině položek.

Příklad 5.4.10. Nad slovem *aabbc* pracuje analyzátor sestrojený v př. 5.4.9 takto (první člen trojice označuje vždy zbytek vstupního slova, druhý člen obsah zásobníku a třetí obsah výstupní pásky):

$$(aabbc; q_0; e) \vdash (abbc; q_0 a q_2; e) \vdash$$
$$\vdash (bbc; q_0 a q_2 a q_7; e) \vdash (bbc; q_0 a q_2 a q_7 A q_{10}; 4) \vdash$$
$$\vdash (bc; q_0 a q_2 a q_7 A q_{10} b q_{11}; 4) \vdash (bc; q_0 a q_2 A q_3; 43) \vdash$$
$$\vdash (c; q_0 a q_2 A q_3 b q_4; 43) \vdash (c; q_0 A q_1; 433) \vdash$$
$$\vdash (e; q_0 A q_1 c q_6; 433) \vdash (e; q_0 A q_1 B q_5; 4336) \vdash$$
$$\vdash \text{slovo je přijato, jeho pravý rozbor je } 43361\,.$$

5.5. LR jazyky a deterministické jazyky

Definice 5.5.1. Jazyk nazveme LR(k) *jazykem*, jestliže existuje LR(k) gramatika, která jej generuje. Řekneme, že jazyk je LR *jazykem*, jestliže je LR(k) jazykem pro nějaké přirozené číslo $k \geqq 0$.

Cílem tohoto článku je dokázat, že libovolný jazyk je LR, právě když je deterministický. Jedna část tvrzení plyne z toho, že LR(k) analyzátory lze simulovat deterministickým zásobníkovým automatem.

Věta 5.5.2. *Každý* LR *jazyk je deterministický.*

Důkaz. V článku 5.2 jsme ukázali, jak převést LR(0) analyzátor na deterministický zásobníkový automat. Jednalo se o způsob obsluhování zásobníku (pozn. 5.2.25) a tentýž způsob lze využít při přechodu od LR(k) analyzátoru k deterministickému zásobníkovému automatu pro libovolné $k \geqq 0$. V případě $k \geqq 1$ vyvstává ještě problém, jak nahradit možnost prohlížení k symbolů dopředu na vstupní pásce, kterou má LR(k) analyzátor. V možnostech deterministického automatu je uchovávat informaci o k-tici symbolů v konečné paměti řídicí jednotky. Deterministický automat tedy může např. provádět manipulaci se zásobníkem se zpožděním až po přečtení potřebných k symbolů. Zde ovšem vzniká další potíž. Kdyby automat pouze udržoval konstantní předstih hlavy, mohlo by se stát, že hlava opustí pravý konec slova dříve, než má automat možnost dokončit simulaci výpočtu. Tomu lze poměrně snadno odpomoci, pokud je pravý konec vstupní pásky označen speciálním koncovým znakem. Na tomto znaku se vstupní hlava zastaví.

Od deterministického automatu s koncovým znakem lze pak přejít k ekvivalentnímu deterministickému zásobníkovému automatu bez koncového znaku, protože podle tabulky 8 jsou deterministické jazyky uzavřeny vůči pravému kvocientu s regulárním jazykem.

Ukážeme, že platí i věta obrácená k větě 5.5.2.

Věta 5.5.3. *Každý deterministický jazyk je* LR(1) *jazyk.*

Důkaz věty je bezprostředním důsledkem následujících tří lemmat.

Lemma 5.5.4. *Nechť L je libovolný bezkontextový jazyk, $\#$ přidaný symbol a $\mathscr{G}$ redukovaná bezkontextová gramatika generující jazyk $L\#$. Jestliže některý stav 1-položkového automatu pro $\mathscr{G}$ obsahuje 1-položku tvaru $(A \to \alpha . \# \beta, \hat{L})$, potom $\hat{L} = \{e\}$ a $1(\beta) = e$.*

Důkaz. Indukcí se snadno ověří, že kdyby existovalo neprázdné slovo $u \in \hat{L}$, nebo $1(\beta) \neq e$, existovalo by odvození

$$S \Rightarrow^* \delta A u v \Rightarrow^* \delta \alpha \# \beta u v \Rightarrow^* w_1 \# w_2$$

pro nějaké $w_2 \neq e$, což je spor s předpokladem, že $w_1 \# w_2 \notin L\#$.

Lemma 5.5.5. *Nechť $L\#$ je* LR(0) *jazyk, přičemž symbol $\#$ se nevyskytuje v žádném slově jazyka L. Potom L je* LR(1) *jazyk.*

Důkaz. Nechť $\mathscr{G}$ je LR(0) gramatika taková, že $L(\mathscr{G}) = L\#$. Sestrojme 1-položkový automat $\mathscr{A}$ pro $\mathscr{G}$. Jakmile některý stav q automatu $\mathscr{A}$ obsahuje 1-položku tvaru $(A \to \alpha . \# \beta, \hat{L})$, je podle 5.5.4 $\hat{L} = \{e\}$. Navíc platí, že q neobsahuje žádnou úplnou 1-položku. Jinak by odpovídající stav položkového automatu obsahoval jednak $A \to \alpha . \# \beta$, jednak úplnou položku, a to by bylo ve sporu s předpokladem, že $\mathscr{G}$ je LR(0) gramatika. Podobně q neobsahuje žádnou další 1-položku s terminálem bezprostředně za tečkou.

Sestrojme nyní gramatiku $\mathscr{G}'$, která se od $\mathscr{G}$ bude lišit pouze tím, že $\#$ bude v $\mathscr{G}'$ patřit mezi neterminály a oproti $\mathscr{G}$ bude mít navíc pravidlo $\# \to e$.

Okamžitě je zřejmé, že $L(\mathscr{G}') = L$.

Z konstrukce 1-položkového automatu plyne, že 1-položkový automat pro $\mathscr{G}'$ se od 1-položkového automatu pro $\mathscr{G}$ liší pouze ve stavech, které u gramatiky $\mathscr{G}$ obsahovaly 1-položku typu $(A \to \alpha . \# \beta, e)$. Odpovídající stav u $\mathscr{G}'$ vždy navíc obsahuje 1-položku $(\# \to ., e)$. Ostatní stavy se shodují. Z toho je vidět, že i 1-položkový automat pro $\mathscr{G}'$ splňuje podmínky z definice LR(1) gramatiky. Stavy, ve kterých je nyní navíc 1-položka $(\# \to ., e)$, už u gramatiky $\mathscr{G}$ neobsahovaly jinou úplnou 1-položku. Pokud takový stav obsahuje nějakou 1-položku $(X \to \gamma . x\delta, L)$ s terminálem x, potom kolize s 1-položkou $(\# \to ., e)$ opět nenastává, protože

$$1(x\delta . \hat{L}) = \{x\} \text{ a } \{x\} \cap \{e\} = \emptyset .$$

[Samozřejmě nenastává ani kolize s $(A \to \alpha . \# \beta, e)$, protože $\#$ je v $\mathscr{G}'$ neterminálem.]

Lemma 5.5.6. *Jestliže L je deterministický jazyk a $\#$ nově přidaný symbol, potom $L\#$ je* LR(0) *jazyk.*

Důkaz. Jazyk $L\#$ je podle 4.4.11 rozpoznatelný deterministickým zásobníkovým automatem pomocí prázdného zásobníku. Proto je podle 5.3.1 LR(0) jazyk.

Důkaz věty 5.5.3. Buď L libovolný deterministický jazyk a $\#$ nějaký nově přidaný symbol. Potom $L\#$ je podle 5.5.6 LR(0) jazyk, a tedy L je podle 5.5.5 LR(1) jazyk.

Věta 5.5.7. *Jazyk je deterministický, právě když je to* LR *jazyk.*

Důkaz plyne bezprostředně z vět 5.5.2 a 5.5.3.

Důsledek 5.5.8. *Každý* LR *jazyk je* LR(1) *jazyk.*

Důkaz. Každý LR jazyk je deterministický podle 5.5.2, a tedy jej podle 5.5.5 generuje vhodná LR(1) gramatika.

Poznámka 5.5.9. Z 5.5.7 pochopitelně neplyne, že každá LR gramatika je LR(1). Naopak, lze ukázat, že pro každé $k > 0$ existuje LR(k) gramatika, která není LR($k-1$).

5.6. LL(k) gramatiky

Zobecněním pojmu LL(1) gramatiky dostaneme pojem LL(k) gramatiky.

Definice 5.6.1. Budiž $k \geqq 1$ přirozené číslo. O bezkontextové gramatice $\mathscr{G} = (\Pi, \Sigma, S, P)$ říkáme, že je LL(k) *gramatika*, jestliže pro libovolná dvě pravidla $A \to \alpha$, $A \to \beta$ $(\alpha \neq \beta)$ gramatiky a libovolnou levou větnou formu $uA\eta$ $(u \in \Sigma^*)$ platí, že

$$k(\alpha\eta) \cap k(\beta\eta) = \emptyset .$$

Čtenář, který už si osvojil, jak souvisí levé derivace s analýzou shora, dokáže jistě odhadnout dosah této definice pro analýzu. Levá větná forma $uA\eta$ odpovídá situaci, kdy v zásobníku je uloženo slovo $A\eta$ (levý konec slova zde opět znamená vrchol zásobníku). Jestliže je splněna podmínka z definice LL(k) gramatiky, znamená to, že k-tice terminálních symbolů, která se postupně objeví na vrcholu zásobníku $\alpha\eta$ (tj. zásobníku po přepsání podle pravidla $A \to \alpha$), se v žádném případě nemůže objevit na vrcholu při zpracování zásobníku $\beta\eta$ a obráceně. Klíč k určení, které z pravidel $A \to \alpha$, $A \to \beta$ použít, tedy lze najít v následující k-tici vstupních symbolů.

Definice 5.6.2. Budiž $k \geq 1$ přirozené číslo. *Silnou* LL(k) *gramatikou* nazveme každou bezkontextovou gramatiku

$$\mathcal{G} = (\Pi, \Sigma, S, P)$$

splňující tuto podmínku: Pro libovolná dvě pravidla $A \to \alpha$, $A \to \beta$ $(\alpha \neq \beta)$ gramatiky a libovolné levé větné formy $uA\eta$, $v A \vartheta$ $(u, v \in \Sigma^*)$ je

$$k(\alpha\eta) \cap k(\beta\vartheta) = \emptyset .$$

Poznámka 5.6.3. Než vyjasníme zdánlivý nesoulad mezi definicí LL(1) gramatiky z čl. 5.1 a definicemi 5.6.1, 5.6.2, uvědomme si, že požadavek kladený na silnou LL(k) gramatiku je silnější než požadavek kladený na LL(k) gramatiku. Proto je každá silná LL(k) gramatika také LL(k) gramatikou. Obráceně to platit nemusí (pro $k \geq 2$), jak ukážeme.

Příklad 5.6.4. Přesvědčíme se, že gramatika

$$S \to 0A00 \mid 1A10 ,$$

$$A \to 1 \mid e$$

je LL(2), ale není to silná LL(2) gramatika. S-pravidla mají na začátku pravých stran odlišné terminály, takže stačí prozkoumat pouze A-pravidla. Existují pouze dvě levé formy obsahující A, a to $0A00$ a $1A10$. Ověřme pro obě podmínku z definice 5.6.1: $2(100) \cap 2(e00) = \emptyset$ a $2(110) \cap 2(e10) = \emptyset$, gramatika je tedy LL(2).

Na druhé straně $2(100) \cap 2(e10) = \{10\}$, což znamená, že gramatika není silná LL(2) gramatika.

Poznámka 5.6.5. Uchýlíme-li se opět k interpretaci definice 5.6.2 v termínech analýzy shora, je možné silné LL(k) gramatiky charakterizovat jako takové, u nichž přepsání symbolu A na vrcholu zásobníku řetězem α při nějakém výpočtu vede k postupnému vydání k-tice terminálních symbolů odlišných od k-tic, které se objeví po přepsání A na β i při libovolném jiném výpočtu. Díky tomu je návrh analyzátorů pro silné LL(k) gramatiky jednodušší než v obecném LL(k) případě.

Následující tvrzení ukazuje, že definice 5.6.1, 5.6.2 jsou v souladu s definicí LL(1) gramatiky, kterou jsme zavedli v čl. 5.1.

Věta 5.6.6. *Gramatika je silná* LL(1) *gramatika, právě když je* LL(1) *gramatika.*

Důkaz. Přímo z definic 5.6.1 a 5.6.2 plyne, že silná LL(1) gramatika je LL(1).

Obráceně předpokládejme, že $\mathscr{G} = (\Pi, \Sigma, S, P)$ je LL(1) gramatika, $A \to \alpha$, $A \to \beta$ libovolná její dvě různá pravidla a $uA\eta$, $vA\vartheta$ nějaké její dvě levé větné formy. Máme ukázat, že

$$1(\alpha\eta) \cap 1(\beta\vartheta) = \emptyset .$$

Předpokládejme opak, tj.

$$a \in 1(\alpha\eta) \cap 1(\beta\vartheta)$$

pro nějaké

$$a \in \Sigma \cup \{e\} .$$

Zřejmě platí, že

$$1(\alpha\eta) = 1(1(\alpha) . 1(\eta))$$

a

$$1(\beta\vartheta) = 1(1(\beta) . 1(\vartheta)) .$$

Rozeberme možné případy.

a) Nechť $e \in 1(\alpha) \cap 1(\beta)$, tzn. $\alpha \Rightarrow^* e$ a $\beta \Rightarrow^* e$. Potom ovšem např.

$$1(\alpha\eta) \cap 1(\beta\eta) \supseteq 1(\eta) ,$$

a tedy

$$1(\alpha\eta) \cap 1(\beta\eta) \neq \emptyset ,$$

což je spor s předpokladem, že $\mathscr{G}$ je LL(1) gramatika. Musí proto buď $e \notin 1(\alpha)$, nebo $e \notin 1(\beta)$.

b) Nechť $e \notin 1(\alpha)$ & $e \notin 1(\beta)$. Potom existuje $a \in \Sigma$ takové, že

$$a \in 1(\alpha) \cap 1(\beta)$$

a opět dostáváme

$$1(\alpha\eta) \cap 1(\beta\eta) \neq \emptyset,$$

a to je spor.

c) Nechť

$$e \notin 1(\alpha),\ e \in 1(\beta)\ \&\ 1(\alpha) \cap 1(\vartheta) \neq \emptyset.$$

Potom také

$$1(\alpha\vartheta) \cap 1(\beta\vartheta) \neq \emptyset,$$

což je spor s definicí LL(1) gramatiky.

d) Zbývající případ

$$e \in 1(\alpha)\ \&\ e \notin 1(\beta)\ \&\ 1(\eta) \cap 1(\beta) \neq \emptyset$$

vede ke sporu analogicky jako c.

Tím jsme dovedli předpoklad, že

$$1(\alpha\eta) \cap 1(\beta\vartheta) \neq \emptyset,$$

ke sporu.

Obraťme nyní pozornost k otázce, jak analyzovat shora obecné LL(k) gramatiky. Je zřejmé, že v každém kroku analýzy je následující krok jednoznačně určen obsahem zásobníku $A\eta$ a následující k-tici vstupních symbolů u. Podle definice 5.6.1 LL(k) gramatiky totiž existuje nejvýše jedno pravidlo $A \to \alpha$ takové, že $u \in k(\alpha\eta)$.

K velkému zjednodušení celého problému přispívá následující tvrzení, jehož důkaz je zřejmý.

Tvrzení 5.6.7. *Nechť $\mathscr{G} = (\Pi, \Sigma, S, P)$ je bezkontextová gramatika. Pro libovolné dva řetězy $v, \omega \in (\Pi \cup \Sigma)^*$ a libovolné $k \geqq 0$ platí:*

$$k(v\,.\,\omega) = k(v\,.\,k(\omega)) = k(k(v)\,.\,k(\omega)).$$

Díky tomuto tvrzení lze podmínku

$$k(\alpha\eta) \cap k(\beta\eta) = \emptyset$$

z definice 5.6.1 nahradit podmínkou

$$k(\alpha \,.\, L) \cap k(\beta \,.\, L) = \emptyset ,$$

kde $L = k(\eta)$. L je konečný, protože $L \subseteq \Sigma^{*k}$.

K rozhodnutí, jaký bude další krok analýzy LL(k) gramatiky, tedy stačí znát následující k-tici symbolů u, vrchní symbol A v zásobníku a jazyk $L = k(\eta)$, kde $A\eta$ je obsah zásobníku. Existuje jen konečně mnoho jazyků $L \subseteq \Sigma^{*k}$, a proto také jen konečně mnoho dvojic tvaru (A, L), kde A je neterminál a $L \subseteq \Sigma^{*k}$. Díky tomu můžeme množinu těchto dvojic považovat za konečnou abecedu, tzn. (A, L) budeme chápat jako jediný symbol abecedy.

A nyní už jsme schopni obecně popsat analyzátor pro danou LL(k) gramatiku $\mathscr{G}$. Analyzátor bude opět sestávat z

jednocestné vstupní pásky,
konečné řídicí jednotky,
zásobníku,
jednocestné výstupní pásky.

Vstupní abeceda bude Σ, výstupními symboly budou čísla pravidel z P. Zásobníková abeceda bude

$$\Gamma = \Sigma \cup \{(X, L);\ X \in \Pi, L \subseteq \Sigma^{*k}\} .$$

Analyzátor bude krok po kroku sledovat úspěšný výpočet nedeterministického automatu rozpoznávajícího $L(\mathscr{G})$ prázdným zásobníkem s tou změnou, že místo každého neterminálu X bude do zásobníku ukládat (X, L) pro vhodné L. Přesněji: jestliže během úspěšného výpočtu nedeterministický zásobníkový automat přečte vstupní úsek u a zásobník v tu chvíli obsahuje řetěz

$$u_1 X_1 u_2 \ldots u_n X_n u_{n+1}$$

pro nějaká $X_1, \ldots, X_n \in \Pi$ a $u_1, \ldots, u_{n+1} \in \Sigma^*$, bude mít LL($k$)-analyzátor v tomtéž kroku rovněž přečten vstupní úsek u a jeho zásobník bude obsahovat řetěz

$$u_1 (X_1, L_1)\, u_2 \ldots u_n (X_n, L_n)\, u_{n+1} ,$$

kde

(*) $L_i = k(u_{i+1}X_{i+1} \dots X_n u_{n+1})$ pro každé i, $1 \leqq i \leqq n$.

Popíšeme, jakým způsobem se této vlastnosti docílí.

1. Začátek výpočtu: v zásobníku analyzátoru je uložen pouze počáteční zásobníkový symbol $(A, \{e\})$. Takový obsah zásobníku pochopitelně splňuje podmínku (*).

2. Obecná situace: zásobník obsahuje řetěz

$$u_1(X_1, L_1)\, u_2 \dots u_n(X_n, L_n)\, u_{n+1}$$

splňující (*), zbývá přečíst vstupní posloupnost v.

a) $u_1 \neq e$: v tomto případě se v následujícím kroku provádí krácení prvního symbolu slova u_1 proti dalšímu vstupnímu symbolu z úseku v. Tím není úsek

$$(X_1, L_1)\, u_2 \dots u_n(X_n, L_n)\, u_{n+1}$$

vůbec zasažen, takže podmínka (*) je splněna i v nově vzniklé konfiguraci.

b) $u_1 = e$: v tomto případě je třeba přepsat (X_1, L_1). Nechť $X_1 \to \alpha_1, \dots, X_1 \to \alpha_r$ jsou všechna X_1-pravidla gramatiky $\mathscr{G}$. Podle předpokladu, že $\mathscr{G}$ je LL(k) gramatika, existuje nejvýše jedno α_i takové, že $k(v) \in k(\alpha_i L_1)$. Pokud takové α_i neexistuje, končí výpočet neúspěšně. Pokud existuje, provedeme přepsání (X_1, L_1) podle pravidla $X_1 \to \alpha_i$ následujícím způsobem. Jestliže $\alpha_i \in \Sigma^*$, stačí zřejmě nahradit (X_1, L_1) řetězem α_i a nově vzniklý obsah zásobníku opět splňuje podmínku (*). Zajímavější je případ, kdy

$$\alpha_i = z_1 Y_1 z_2 \dots z_s Y_s z_{s+1}$$

pro nějaká

$$Y_1, \dots, Y_s \in \Pi\,,\ z_1, \dots, z_{s+1} \in \Sigma^*,\ s \geqq 1\,.$$

Zde se symbol (X_1, L_1) nahradí řetězem

$$\hat{\alpha}_i = z_1(Y_1, \hat{L}_1)\, z_2 \dots z_s(Y_s, \hat{L}_s)\, z_{s+1}$$

tak, že

$$(**) \qquad \hat{L}_j = k(z_{j+1} \dots Y_s z_{s+1} u_2 \dots u_n X_n u_{n+1})$$

pro každé j, $1 \leqq j \leqq s$,

aby nově vzniklý obsah zásobníku opět splňoval podmínku $(*)$. Připomeňme ještě, že

$$k(u_2 \dots u_n X_n u_{n+1}) = L_1$$

a že tedy podle 5.6.7 je

$$(***) \qquad \hat{L}_j = k(z_{j+1} \dots Y_s z_{s+1} \,.\, L_1) \text{ pro každé } j,\ 1 \leqq j \leqq s.$$

Tím jsme ověřili, že při popsaném postupu obsah zásobníku stále zachovává vlastnost $(*)$, a navíc jsme zjistili, že výpočet probíhá deterministicky, neboť v každé situaci existuje nejvýše jedno pokračování [díky tomu, že výchozí gramatika je LL(k)].

Zatím jsme ještě nevyjasnili otázku, jak realizovat dvě klíčové operace, které je během této analýzy třeba provádět:

1. rozhodnutí, podle kterého pravidla $X_1 \to \alpha_i$ se má přepsat vrchní symbol (X_1, L_1);
2. nalezení řetězu $\hat{\alpha}_i$.

ad 1: pro každou dvojici (X, L) lze definovat parciální funkci

$$f_{X,L}: \Sigma^{*k} \to \{\alpha_1, \dots, \alpha_r, \perp\}$$

(kde $X \to \alpha_1, \dots, X \to \alpha_r$ jsou všechna X-pravidla) takto:

$$f_{X,L}(u) = \begin{cases} \alpha_i, \text{ jestliže } u \in k(\alpha_i \,.\, L) \text{ a } u \notin k(\alpha_j \,.\, L) \\ \quad \text{pro všechna ostatní } \alpha_j, \\ \text{nedefinováno, jestliže pro žádné } \alpha_i \text{ není } u \in k(\alpha_i \,.\, L), \\ \perp, \text{ jestliže } u \in k(\alpha_i \,.\, L) \cap k(\alpha_j \,.\, L) \\ \quad \text{pro nějaká } \alpha_i \neq \alpha_j. \end{cases}$$

Každou takovou funkci lze zadat ve formě tabulky. Operaci 1 proto lze vyřešit sestavením tabulek pro všechna (X, L) a jejich uložením do konečné paměti řídicí jednotky. Je zřejmé, že funkce $f_{X,L}$, které někde nabývají hodnoty $\perp$, není třeba zařazovat, protože taková dvojice (X, L) se během LL(k)-analýzy nemůže vyskytnout.

má β_i příslušné α_i je podle (***) jednoznačně určeno řetězem u_i a jazykem L_1. Opět tedy existuje funkce, která libovolné pravé straně nějakého pravidla a libovolnému $L \subseteq \Sigma^{*k}$ přiřazuje příslušný modifikovaný řetěz. Tuto funkci lze opět uložit v paměti řídicí jednotky jako tabulku.

K sestavení tabulek obou typů je třeba vědět, jakým způsobem se počítají hodnoty $k(\alpha)$, resp. $k(L)$ pro $\alpha \in (\Sigma \cup \Pi)^*$ a konečný jazyk $L \subseteq \Sigma^*$. To nyní ukážeme.

Konstrukce 5.6.8. Konstrukce $k(u)$ pro $u \in \Sigma^*$: $k(u)$ je rovno prefixu délky k z u (resp. u, pokud $|u| \leqq k$).

Konstrukce $k(L)$ pro konečný jazyk $L \subseteq \Sigma^*$:

$$k(L) = \{k(u);\ u \in L\}\,.$$

Konstrukce $k(X)$ pro $X \in \Pi$: Pro všechna $x \in \Sigma \cup \Pi$ postupně sestrojujeme množiny $k_0(x), k_1(x), \ldots$, takto:

1. položme $k_i(x) = \{x\}$ pro všechna $i \geqq 0$, jakmile $x \in \Sigma$.
2. $k_0(X) = \{u \in \Sigma^*;\ |u| < k\ \&\ X \to u \in P\} \cup$
$\cup\ \{u \in \Sigma^*;\ |u| = k\ \&\ X \to u\alpha \in P,$ pro nějaké $\alpha\}$

pro každé $X \in \Pi$.

3. $k_{i+1}(X) = k_i(X) \cup \{u;\ u \in k(k_i(y_1)\,.\,k_i(y_2) \ldots k_i(y_n))$
pro nějaké $X \to y_1 \ldots y_n \in P\}$

pro každé $X \in \Pi$.

Jelikož $k_i(x) \subseteq k_{i+1}(x) \subseteq \Sigma^{*k}$ pro všechna x, i, existuje i_0 tak, že $k_{i_0}(x) \quad k_{i_0+1}(x)$ současně pro všechna $x \in \Pi \cup \Sigma$. Pro každé $X \in \Pi$ položíme $k(X) = k_{i_0}(X)$.

Konstrukce $k(\alpha)$ pro $\alpha = x_1 \ldots x_n$, $x_i \in \Pi \cup \Sigma$:

$$k(x_1 x_2 \ldots x_n) = k(k(x_1)\,.\,k(x_2) \ldots k(x_n))\,.$$

Jazyk $k(x_1) \ldots k(x_n) \subseteq \Sigma^*$ je konečný, a proto je tato konstrukce realizovatelná podle předchozích bodů.

Důkaz správnosti popsaných konstrukcí přenecháváme čtenáři.

6. ALGORITMICKY NEŘEŠITELNÉ PROBLÉMY

ZÁKLADNÍ ČÁST

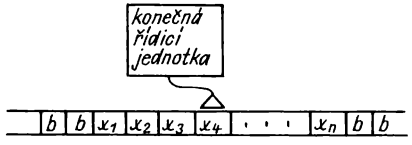
Teoretický aparát předchozích kapitol umožňuje odpovědět na mnohé otázky týkající se vlastností automatů a gramatik. Tak například na základě věty 2.2.14 dovedeme pro libovolné dva konečné automaty rozhodnout, zda reprezentují tentýž jazyk. Důležité je, že tento problém dovedeme řešit algoritmicky, tzn. existuje algoritmus, který pro libovolné dva konečné automaty nad danou abecedou vydá po konečném počtu kroků odpověď, zda automaty jsou ekvivalentní či nikoliv. Podobně pro každou bezkontextovou gramatiku umíme algoritmicky rozhodnout (viz 4.1.2), zda generuje prázdný jazyk.

V této kapitole se zaměříme na problémy, které algoritmicky rozhodnutelné nejsou.

6.1. Turingovy stroje a jazyky typu 0

Základní model Turingova stroje je schematicky znázorněn na obr. 45.

Takovýto Turingův stroj se skládá z konečné řídicí jednotky a nekonečné pásky rozdělené na jednotlivá políčka. Řídicí jed-



Obr. 45

notka se může dostávat do konečně mnoha různých stavů a v každém políčku pásky může být zapsán symbol z konečné abecedy stroje nebo políčko může být prázdné. V každém okamžiku má stroj přístup k jedinému políčku pásky. Po jeho uplynutí může

stroj změnit obsah čteného políčka, posunout se o jedno políčko doprava nebo doleva a změnit stav řídicí jednotky. Všechny tyto změny závisí jednak na obsahu čteného políčka, jednak na vnitřním stavu řídicí jednotky.

Uveďme formální definici.

Definice 6.1.1. *Turingův stroj* je pětice $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$, kde Q a Σ jsou po řadě konečné neprázdné množiny stavů a symbolů stroje, $\delta: (Q - F) \times \Sigma \to Q \times \Sigma \times \{-1, +1\}$ je parciální přechodová funkce, $q_0 \in Q$ je počáteční stav a $F \subseteq Q$ je cílová množina (množina koncových stavů).

Jestliže $\delta(q, x) = (q', x', d)$, $d \in \{-1, +1\}$, znamená to, že stroj v každé situaci, kdy jeho vnitřní stav je q a nachází se na políčku se symbolem x, přepíše obsah políčka na symbol x', přejde do stavu q' a potom se posune o d políček doprava.

Budeme předpokládat, že abeceda Σ každého Turingova stroje obsahuje jistý symbol b. Jestliže na políčku je symbol b, budeme to interpretovat tak, že políčko je prázdné.

Definice 6.1.2. *Konfigurací Turingova stroje*

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

budeme nazývat každé slovo tvaru uqv, kde $u, v \in \Sigma^*$, $q \in Q$.

Konfigurace uqv bude odpovídat situaci, kdy na pásce stroje je napsáno slovo uv, ostatní políčka jsou prázdná, stroj je ve vnitřním stavu q a čte první (odleva) symbol slova v.

Poznámka 6.1.3. Konfigurace uqv odpovídá stejné situaci Turingova stroje jako libovolná z konfigurací $b^i uqv b^j$, kde $i, j \geq 0$. Proto je často možné takové konfigurace ztotožňovat. Speciálně konfiguraci tvaru qv je pohodlné ztotožňovat s bqv a konfiguraci tvaru uq s uqb.

Definice 6.1.4. Budiž $K = uyqxv$, kde $u, v \in \Sigma^*$, $x, y \in \Sigma$ a $q \in Q$, konfigurace Turingova stroje $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$. Potom *konfigurací bezprostředně následující po konfiguraci* K nazveme konfiguraci

$$uyx'q'v, \text{ jestliže } \delta(q, x) = (q', x', +1)$$

a

$$uq'yx'v\,,\ \text{jestliže}\ \delta(q,x) = (q', x', -1)\,.$$

Jestliže konfigurace K_2 bezprostředně následuje po K_1, píšeme $K_1 \vdash_{\mathscr{M}} K_2$. Relace $\vdash^*_{\mathscr{M}}$ je definována jako reflexívní a tranzitívní uzávěr relace $\vdash_{\mathscr{M}}$. Jestliže $K \vdash^*_{\mathscr{M}} K'$, říkáme, že K' následuje po K nebo že stroj $\mathscr{M}$ se z K dostane do K' apod. *Počáteční konfigurací* nazýváme každou konfiguraci tvaru $q_0 w$, kde

$$w \in (\Sigma - \{b\})^*.$$

Koncovou konfigurací nazýváme konfiguraci tvaru uqv, kde u, $v \in \Sigma^*$, $q \in F$.

Říkáme, že *Turingův stroj* $\mathscr{M}$ *přijímá slovo* $w \in (\Sigma - \{b\})^*$, jestliže $q_0 w \vdash^*_{\mathscr{M}} K$ pro nějakou koncovou konfiguraci K.

Jazyk rozpoznávaný strojem $\mathscr{M}$ je jazyk

$$L(\mathscr{M}) = \{w \in (\Sigma - \{b\})^*;\ \mathscr{M}\ \text{přijímá}\ w\}\,.$$

Poznámka 6.1.5. Podle poznámky 6.1.3 je $qv \vdash K$, jakmile $bqv \vdash K$ a $uq \vdash K$, jakmile $uqb \vdash K$.

Příklad 6.1.6. Sestrojme Turingův stroj

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

rozpoznávající jazyk

$$L = \{w;\ w \in \{c, d\}^* \ \&\ w = w^R\}\,,$$

tj. jazyk tvořený symetrickými slovy v abecedě $\{c, d\}$. Stroj $\mathscr{M}$ bude vypadat takto:

$$\Sigma = \{b, c, d\}\,,$$
$$Q = \{q_0, q_1, q_c, q_d, q_1^c, q_1^d, q_f\}\,,$$
$$F = \{q_f\}$$

a

I. $\delta(q_0, c) = (q_c, b, +1)\,,$
$\delta(q_0, d) = (q_d, b, +1)\,;$

II. $\delta(q_c, c) = (q_c, c, +1)\,,$
$\delta(q_c, d) = (q_c, d, +1)\,,$

$$\delta(q_d, c) = (q_d, c, +1),$$
$$\delta(q_d, d) = (q_d, d, +1);$$

III. $\delta(q_c, b) = (q_1^c, b, -1),$
$$\delta(q_d, b) = (q_1^d, b, -1);$$

IV. $\delta(q_1^c, c) = (q_1, b, -1),$
$$\delta(q_1^d, d) = (q_1, b, -1),$$
$$\delta(q_1, c) = (q_1, c, -1),$$
$$\delta(q_1, d) = (q_1, d, -1);$$

V. $\delta(q_1^c, b) = (q_f, b, +1) = \delta(q_1^d, b);$

VI. $\delta(q_1, b) = (q_0, b, +1);$

VII. $\delta(q_0, b) = (q_f, b, +1).$

Způsob, jakým stroj $\mathcal{M}$ rozpoznává jazyk L, je celkem zřejmý. Když ve stavu q_0 přečte první symbol slova, smaže jej a ve své konečné paměti uchovává informaci o tom, který symbol to byl (pravidla I). Poté postupuje doprava, dokud nenajde konec slova (pravidla II). Jakmile najde konec slova, vyhledá poslední symbol (III). Jestliže tento symbol souhlasí s informací uloženou v konečné paměti, vymaže políčko a postupuje doleva, dokud nenajde začátek slova (IV). Potom vyhledá začátek slova a celá procedura začíná znovu (VI). Jestliže někdy první symbol slova nesouhlasí s posledním, dostane se $\mathcal{M}$ do situace, že je ve stavu q_1^c a čte d, nebo ve stavu q_1^d a čte c, tj. do situace, kdy další krok není určen, tzn. slovo není přijato, protože ani q_1^c, ani q_1^d nejsou koncové stavy. Do koncového stavu se stroj dostane pouze probíhá-li proces úspěšně tak dlouho, až se páska vyprázdní (V a VII). Také tehdy se stroj zastaví.

Např. slovo cdc je přijato výpočtem:

$$q_0cdc \vdash bq_cdc \vdash dq_cc \vdash dcq_c \vdash dq_1^cc \vdash q_1dd \vdash$$
$$\vdash q_1bd \vdash q_0d \vdash q_d \vdash q_1^d \vdash q_f.$$

V zápisu výpočtu jsme užili úspornějšího zápisu ve smyslu poznámky 6.1.3.

O možnostech gramatik typu 0 dává představu následující věta.

Věta 6.1.7. *Každý jazyk rozpoznatelný Turingovým strojem je typu* 0.

Důkaz. Nechť $\mathscr{M} = (Q, \Sigma, \delta, q_0, F)$ je Turingův stroj. Sestrojíme gramatiku $\mathscr{G} = (\Pi, \Sigma', S, P)$ takovou, že $L(\mathscr{G}) = L(\mathscr{M})$.

K abecedě Σ sestrojíme abecedu dvojníků $\bar{\Sigma}$. Dvojníka symbolu $x \in \Sigma$ budeme označovat $\bar{x}$. Gramatiku $\mathscr{G}$ pak definujeme takto:

$$\Sigma' = \Sigma - \{b\}, \quad \Pi = Q \cup \bar{\Sigma} \cup \{S, A, B, C\}.$$

Množina P obsahuje tato pravidla:

I. $S \to Aq_0B$,

$A \to x A \bar{x}$ pro všechna $x \in \Sigma - \{b\}$,

$A \to B$,

$B \to \bar{b}B \mid \bar{b}$;

II. $\bar{x}q\bar{y} \to q'\bar{x}'\bar{y}$ pro všechna $y \in \Sigma$,

jestliže $\delta(q, x) = (q', x', +1)$,

$\bar{x}q\bar{y} \to \bar{x}'\bar{y}q'$ pro všechna $y \in \Sigma$,

jestliže $\delta(q, x) = (q', x', -1)$;

III. $q \to C$ pro všechna $q \in F$,

$Ca \to C$ pro všechna $a \in \bar{\Sigma}$,

$aC \to C$ pro všechna $a \in \bar{\Sigma}$,

$C \to e$.

Ukažme, proč $L(\mathscr{G}) = L(\mathscr{M})$. Všimněme si, že levé strany pravidel skupin II a III neobsahují žádný ze symbolů S, A, B a že tyto symboly nemohou vzniknout aplikací pravidel těchto skupin. Jestliže tedy v nějaké derivaci je aplikace pravidel typu I předcházena aplikací pravidla typu II nebo III, týkají se tyto dvě aplikace disjunktních úseků slova, a proto lze jejich pořadí vyměnit, aniž se tím ovlivní výsledek. Takovýmito postupnými výměnami je možné dosáhnout toho, že nejprve jsou aplikována pravidla typu I a potom pravidla zbývajících dvou skupin. Z tvaru

pravidel těchto dvou skupin je však okamžitě zřejmé, že po první aplikaci pravidla typu III už není možné použít žádné pravidlo skupiny II.

Jestliže $w \in L(\mathscr{G})$, a tedy $S \Rightarrow_{\mathscr{G}}^{*} w$, $w \in \Sigma'^{*}$, lze bez újmy na obecnosti předpokládat, že v derivaci slova se nejprve vyskytovaly aplikace pravidel skupiny I, potom skupiny II a nakonec skupiny III.

Aplikací pravidel skupiny I vznikne slovo tvaru

$$wb^i\bar{w}^{\mathrm{R}}q_0\bar{b}^j \quad \text{pro nějaká } i, j > 0 .$$

Přitom $\bar{w}$ označuje dvojníka slova w, tj. pro $w = x_1 \dots x_n$ je $\bar{w} = \bar{x}_1 \dots \bar{x}_n$.

Pravidla typu II simulují výpočet stroje $\mathscr{M}$ na slově w. Je však simulován zrcadlový obraz výpočtu na dvojnících původních symbolů.

K pravidlům III je možné přejít pouze tehdy, jestliže se v simulovaném výpočtu dostane stroj do koncového stavu, tj. jestliže $w \in L(\mathscr{M})$. V tom případě se ve slově objeví C, tj. slovo je tvaru $w\bar{u}C\bar{v}$, kde $\bar{u}, \bar{v} \in \bar{\Sigma}^{*}$. Symbol C zlikviduje postupně slova $\bar{u}$ a $\bar{v}$ a nakonec sám zmizí. Teprve pak vznikne terminální řetěz, a to w.

Tím jsme dokázali, že $L(\mathscr{G}) \subseteq L(\mathscr{M})$.

Jestliže obráceně $w \in L(\mathscr{M})$, potom pravidly I lze vygenerovat slovo $wb^i w^{\mathrm{R}} q_0 \bar{b}^j$, kde i a j jsou dostatečně velká čísla, aby na úseku pásky $b^j w b^i$ proběhl celý výpočet stroje $\mathscr{M}$ nad w. Potom pravidla II simulují výpočet až do okamžiku, kdy se $\mathscr{M}$ dostane do koncového stavu. Pak se začnou aplikovat pravidla III tak dlouho, až zůstane samotný řetěz w. Tedy $w \in L(\mathscr{G})$ a věta je dokázána.

Příklad 6.1.8. Stroji z př. 6.1.6 odpovídá gramatika daná pravidly

I
$$\begin{aligned} S &\to Aq_0B, \\ A &\to cA\bar{c} \mid dA\bar{d}, \\ A &\to B, \\ B &\to bB \mid \bar{b} ; \end{aligned}$$

II. $\left.\begin{array}{l} \bar{c}q_0\bar{b} \to q_c\bar{b}\bar{b}\,, \\ \bar{c}q_0\bar{c} \to q_c\bar{b}\bar{c}\,, \\ \bar{c}q_0\bar{d} \to q_c\bar{b}\bar{d}\,, \\ \bar{d}q_0\bar{b} \to q_d\bar{b}\bar{b}\,, \\ \bar{d}q_0\bar{c} \to q_d\bar{b}\bar{c}\,, \\ \bar{d}q_0\bar{d} \to q_d\bar{b}\bar{d}\,, \end{array}\right\}$ odpovídá pravidlům I v příkladu 6.1.6

$\left.\begin{array}{l} \bar{b}q_c\bar{b} \to \bar{b}\bar{b}q_1^c\,, \\ \bar{b}\bar{q}_c c \to \bar{b}\bar{c}q_1^c\,, \\ \bar{b}q_c\bar{d} \to \bar{b}\bar{d}q_1^c\,, \\ \ldots \end{array}\right\}$ odpovídá pravidlům III

$\left.\begin{array}{l} \bar{b}q_0\bar{b} \to q_f\bar{b}\bar{b}\,, \\ \bar{b}q_0\bar{c} \to q_f\bar{b}\bar{c}\,, \\ \bar{b}q_0\bar{d} \to q_f\bar{b}\bar{d}\,, \\ \ldots \end{array}\right\}$ odpovídá pravidlům VII

obdobně pro ostatní skupiny pravidel z př. 6.1.6;

III. $q_f \to C\,,$
$C\bar{b} \to C\,,$
$C\bar{c} \to C\,,$
$C\bar{d} \to C\,,$
$\bar{b}C \to C\,,$
$\bar{c}C \to C\,,$
$\bar{d}C \to C\,,$
$C \to e\,.$

Přijímajícímu výpočtu nad slovem cdc odpovídá odvození

$$S \Rightarrow Aq_0B \Rightarrow^* cdcA\bar{c}\bar{d}\bar{c}q_0B \Rightarrow cdcB\bar{c}\bar{d}\bar{c}q_0B \Rightarrow^*$$
$$\Rightarrow^* cdc\bar{b}\bar{c}\bar{d}\bar{c}q_0\bar{b} \Rightarrow cdc\bar{b}\bar{c}\bar{d}q_c\bar{b}\bar{b} \Rightarrow cdc\bar{b}\bar{c}q_c\bar{d}\bar{b}\bar{b} \Rightarrow$$
$$\Rightarrow cdc\bar{b}q_c\bar{c}\bar{d}\bar{b}^2 \Rightarrow cdc\bar{b}\bar{c}q_1^c\bar{d}\bar{b}^2 \Rightarrow cdc\bar{b}\bar{b}\bar{d}q_1\bar{b}\bar{b} \Rightarrow$$
$$\Rightarrow cdc\bar{b}^2\bar{d}\bar{b}q_1\bar{b} \Rightarrow cdc\bar{b}^2\bar{d}q_0\bar{b}^2 \Rightarrow cdc\bar{b}^2q_d\bar{b}^3 \Rightarrow$$
$$\Rightarrow cdc\bar{b}^3q_1^d\bar{b}^2 \Rightarrow cdc\bar{b}^2q_f\bar{b}^3 \Rightarrow$$
$$\Rightarrow cdc\bar{b}^2C\bar{b}^3 \Rightarrow^* cdcC \Rightarrow cdc\,.$$

Vyslovíme ještě větu, která uzavře charakterizaci jazyků typu 0 pomocí Turingových strojů.

Věta 6.1.9. *Každý jazyk typu* 0 *je rozpoznatelný Turingovým strojem.*

Důkaz. Podrobný důkaz této věty je zdlouhavý, i když jeho hlavní myšlenka je poměrně jednoduchá. Spokojíme se tedy s tím, že popíšeme hlavní ideu důkazu.

Pro danou gramatiku $\mathscr{G} = (\Pi, \Sigma, S, P)$ je možné každou derivaci $S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n = w$, $w \in \Sigma^*$, zakódovat jako slovo $\# S \# \alpha_1 \# \ldots \# \alpha_n \#$. Každou derivaci lze proto ztotožnit s jistým slovem v abecedě $\Pi \cup \Sigma \cup \{\#\}$. Je možné sestrojit Turingův stroj $\mathscr{M}_1$ takový, že přijímá právě slova typu $\# \alpha \# \beta \#$, kde $\alpha, \beta \in (\Pi \cup \Sigma)^*$ a $\alpha \Rightarrow_{\mathscr{G}} \beta$. Tento stroj je možné modifikovat tak, že vznikne stroj $\mathscr{M}_2$, který přijímá právě ta slova v abecedě $\Pi \cup \Sigma \cup \{\#\}$, která označují derivace v gramatice $\mathscr{G}$. Konečně lze sestrojit stroj $\mathscr{M}$, který nalevo od vstupního slova w generuje postupně (dokud výpočet neskončí) všechna slova v abecedě $\Pi \cup \Sigma \cup \{\#\}$, počínaje nejkratšími. (To může dělat např. tak, že ztotožní slova v abecedě $\Pi \cup \Sigma \cup \{\#\}$ s q-adickým zápisem čísel, kde $q = |\Pi \cup \Sigma \cup \{\#\}|$ – viz čl. 7.2.) Pokaždé, když takto vygeneruje nějaké slovo, zkontroluje jako $\mathscr{M}_2$, zda se jedná o derivaci v gramatice $\mathscr{G}$. Jestliže ano, ověří, zda poslední člen derivace je w. Řetěz w zůstává stále na pásce. V příznivém případě přejde do koncového stavu a zastaví se. Jinak na místo dosavadního slova vygeneruje slovo následující a pokračuje stejným způsobem. Je zřejmé, že popsaný stroj přijímá právě jazyk $L(\mathscr{G})$.

Důsledek 6.1.10. *Jazyk je typu* 0, *právě když je rozpoznatelný Turingovým strojem.*

Důsledek 6.1.10 zapadá do širšího okruhu výsledků, které svědčí ve prospěch tzv. Turingovy teze, o níž podrobněji pojednáme v kap. 7. Ukazuje také, že gramatiky typu 0 generují právě všechny tzv. rekurzívně spočetné jazyky, na něž se vztahuje množství hlubokých výsledků z podrobně rozpracované teorie rekurzívně spočetných množin. Vynikající monografií věnovanou tomuto tématu je kniha [27].

6.2. Algoritmicky neřešitelné problémy

V definici 6.1.4 jsme požadovali, aby se při přijetí slova Turingův stroj dostal do koncového stavu. Podle definice 6.1.1 není z takového stavu další přechod definován, a výpočet proto v okamžiku přijetí končí. U slov, která nejsou strojem přijata, mohou nastat dvě možnosti:

a) stroj skončí výpočet, aniž se dostal do koncového stavu;

b) stroj nikdy neskončí výpočet.

Má-li Turingův stroj reprezentovat jistý jazyk L tak, že o každém slově rozhodne, zda patří do L nebo ne, je přirozené požadovat, aby konstrukce stroje vylučovala možnost b). To vede k následující definici.

Definice 6.2.1. Řekneme, že *Turingův stroj* $\mathscr{M}$ *rozhoduje jazyk* L v abecedě Σ, jestliže $L \subseteq \Sigma^*$, $L = L(\mathscr{M})$ a pro každé slovo $w \in \Sigma^*$ je výpočet stroje $\mathscr{M}$ nad w konečný.

Jazyky rozhodnutelné Turingovým strojem nazýváme *rekurzívními jazyky*.

Pro vztah rekurzívních jazyků (tj. jazyků rozhodnutelných Turingovým strojem) a rekurzívně spočetných jazyků (tj. jazyků rozpoznatelných Turingovým strojem) platí tato věta:

Věta 6.2.2 (Postova). *Jazyk* $L \subseteq \Sigma^*$ *je rekurzívní, právě když* L *i* $\Sigma^* - L$ *jsou rekurzívně spočetné jazyky.*

Poznámka 6.2.3. Způsob, kterým uvedeme důkaz Postovy věty, je obvyklý v pokročilejších partiích teorie rekurzívních funkcí, např. v již citované knize [27], a předpokládá u čtenáře dostatečné zvládnutí některé konkrétní formalizace pojmu algoritmu. Potom lze zadávat algoritmy nikoli jejich přesným popisem, ale pouze volnějšími formulacemi, na jejichž základě lze příslušný popis sestrojit. Přitom je možné soustředit se na nejdůležitější obraty a nezatěžovat text detailními technickými konstrukcemi. Proto bude důkaz průkazný jen pro čtenáře, který je schopen převést algoritmy, o kterých budeme hovořit, do formálního zápisu.

Čtenářům, kteří znají některý z univerzálních programovacích jazyků, může pomoci, budou-li promýšlet, jak by příslušný algo-

ritmus naprogramovali. Ostatní se mohou s problematikou detailněji seznámit např. v kap. 6 a 7 knihy [20].

Důkaz věty 6.2.2. Jestliže $\mathscr{M}_1$ je stroj rozpoznávající jazyk L a $\mathscr{M}_2$ stroj rozpoznávající jeho doplněk, pak stroj $\mathscr{M}$ rozhodující L získáme takto: pro každé vstupní slovo w bude $\mathscr{M}$ simulovat střídavě výpočet $\mathscr{M}_1$ i $\mathscr{M}_2$ nad w. Protože $\mathscr{M}_1$ a $\mathscr{M}_2$ rozpoznávají komplementární jazyky, právě jeden z nich w přijme. Zaručeně tedy $\mathscr{M}$ po konečném počtu kroků získá informaci, zda $w \in L$, a podle toho vydá odpověď.

Obráceně, jestliže L je rekurzívní jazyk, pak existuje stroj $\mathscr{M}$ rozhodující L. Tento stroj lze prohozením role koncových a nekoncových stavů modifikovat na stroj rozhodující, a tedy i rozpoznávající doplněk jazyka L.

Věta, kterou nyní vyslovíme, ukazuje, že existuje rekurzívně spočetný jazyk, jehož doplněk rekurzívně spočetný není.

Věta 6.2.4. *Existuje rekurzívně spočetný jazyk, který není rekurzívní. Jinými slovy: existuje jazyk rozpoznatelný Turingovým strojem, který není rozhodnutelný žádným Turingovým strojem.*

Poznámka 6.2.5. O úvahách, které použijeme v důkazu věty 6.2.4, platí totéž, co jsme uvedli v poznámce 6.2.3 před důkazem Postovy věty. Porozumění důkazu není k pochopení dalšího textu nutné. Uvádíme jej zde zejména s ohledem na neobyčejně silný vliv, který měla věta 6.2.4 a její důsledky na rozvoj četných partií moderní matematiky.

Důkaz věty 6.2.4. Označme symbolem **M** množinu všech Turingových strojů, které mají abecedu $\{0, 1, b\}$. Lze ukázat, že

(a) každý stroj z **M** je možné zadat jistým slovem v abecedě $\{0, 1\}$.

Abychom ověřili pravdivost tvrzení (a), stačí si uvědomit, že množiny stavů strojů z **M** lze volit jako podmnožiny jisté spočetné množiny $Q = \{q_0, q_1, q_2, \ldots\}$. Každou instrukci stroje z **M** je možné psát ve tvaru

(1) $q_i \; s \; q_j \; s' \; d$,

kde

$$q, q' \in Q,\ x, x' \in \{0, 1, b\},\ d \in \{-1, +1\}.$$

Každý stroj z **M** lze pak zapsat jako posloupnost instrukcí

$$\S i_1 \# i_2 \# \ldots \# i_n \S,$$

kde i_j jsou tvaru $(+)$. Můžeme tedy každý takový stroj zadat jako jisté slovo v nekonečné abecedě

$$\{\S, \#, \rightarrow, 0, 1, b, -1, +1\} \cup Q.$$

Všechny prvky této abecedy lze zakódovat v abecedě $\{0, 1\}$, např. takto:

$$K(\S) = 101,\ K(\#) = 1001, \ldots, K(+1) = 10^8 1,$$
$$K(q_i) = 10^{9+i} 1$$

pro $i = 0, 1, \ldots$. Tím způsobem se tedy každému $\mathscr{M} \in \mathbf{M}$ přiřadí kód $K(\mathscr{M}) \in \{0, 1\}^*$. Navíc pro toto kódování zřejmě platí, že

(b) jazyk

$$L_K = \{w \in \{0, 1\}^*;\ w = K(\mathscr{M}) \text{ pro nějaký stroj } \mathscr{M} \in \mathbf{M}\}$$

je rekurzívní.

Pro každé slovo $w \in \{0, 1\}^*$ lze totiž snadno rozhodnout, zda je kódem $K(u)$ nějakého slova u v abecedě

$$\{\S, \#, \rightarrow, 0, 1, b, -1, +1\} \cup Q$$

a jestliže ano, pak zda je u programem nějakého Turingova stroje $\mathscr{M}$.

Definujme jazyk $L_U \subseteq \{0, 1, 2\}^*$ tímto předpisem:

$$L_U = \{K(\mathscr{M})\, 2w;\ w \in \{0, 1\}^*,$$
$$\mathscr{M} \in \mathbf{M} \text{ a } \mathscr{M} \text{ skončí po konečném počtu kroků výpočet nad } w\}.$$

O tomto jazyku platí následující tvrzení, které uzavírá důkaz věty.

Lemma 6.2.6. *Jazyk L_U je rekurzívně spočetný, ale není rekurzívní.*

Důkaz. (I) L_U je rekurzívně spočetný jazyk.

Vzhledem k tvrzení (b) lze pro libovolné slovo $v \in \{0, 1, 2\}^*$ rozhodnout, zda je tvaru $K(\mathscr{M})\, 2w$ pro nějaké $\mathscr{M} \in \mathbf{M}$ a $w \in \{0, 1\}^*$. Pro slova tohoto typu je pak možné na základě kódu $K(\mathscr{M})$ simulovat výpočet stroje $\mathscr{M}$ nad w. Jestliže výpočet skončí, je v přijato. Čtenář znalý některého formalismu pro popis všech algoritmů se může přesvědčit, že uvedený postup lze ve zvoleném formalismu naprogramovat. Speciálně existuje Turingův stroj rozpoznávající L_U.

V programátorské terminologii je výsledný stroj možné specifikovat jako interpret programů pro stroje z $\mathbf{M}$.

(II) L_U není rekurzívní.

Kdyby byl jazyk L_U rekurzívní, byl by rekurzívní také jazyk

$$L_d = \{w;\ w = K(\mathscr{M}) \text{ pro nějaký } \mathscr{M} \in \mathbf{M},$$
$$\text{který má navíc vlastnost, že po konečném počtu}$$
$$\text{kroků skončí výpočet nad } K(\mathscr{M})\}\,.$$

Pro libovolné $w \in \{0, 1\}^*$ totiž platí toto:

$$w \in L_d \Leftrightarrow w2w \in L_U\,.$$

Algoritmus pro rozhodování jazyka L_U bychom tedy snadným způsobem modifikovali na algoritmus rozhodující L_d. Abychom ověřili tvrzení (II), stačí tedy dokázat, že L_d není rekurzívní. Důkaz provedeme sporem.

Kdyby byl L_d rekurzívní, byl by podle 6.2.2 rekurzívní také jeho doplněk $\bar{L}_d = \{0, 1\}^* - L_d$, a tedy také jazyk

$$L_{\text{neg}} = \{w;\ w = K(\mathscr{M}) \text{ pro nějaký } \mathscr{M} \in \mathbf{M}$$
$$\text{a výpočet } \mathscr{M} \text{ nad } K(\mathscr{M}) \text{ je nekonečný}\}$$

by byl rekurzívní, neboť pro libovolné $w \in \{0, 1\}^*$

$$w \in L_{\text{neg}} \Leftrightarrow w \in L_K \,\&\, w \in \bar{L}_d\,.$$

Algoritmus rozhodující L_{neg} bychom získali rozhodováním příslušnosti slova k L_K a $\bar{L}_d$. Z předpokladu rekurzivity L_d jsme odvodili rekurzivitu L_{neg}. Budiž tedy $\mathscr{M}$ Turingův stroj rozhodující L_{neg}. Stroj $\mathscr{M}$ může mít kromě symbolů $\{0, 1, b\}$ ještě ko-

nečný počet dalších pracovních symbolů. Podobným obratem jako při zavedení kódování K lze zakódovat abecedu stroje $\mathscr{M}$ pomocí jistých k-tic symbolů z $\{0, 1\}$. Stroj $\mathscr{M}$ pak lze transformovat na jistý $\mathscr{M}' \in \mathbf{M}$ rovněž rozhodující L_{neg}. Tento stroj je možné dále modifikovat na jistý stroj $\hat{\mathscr{M}} \in \mathbf{M}$ tak, že pro libovolné $w \in \{0, 1\}^*$

$$\hat{\mathscr{M}} \text{ skončí výpočet nad } w \Leftrightarrow \mathscr{M}' \text{ přijímá } w \Leftrightarrow w \in L_{\text{neg}}.$$

Z vlastností $\hat{\mathscr{M}}$ pak plyne, že

$$\hat{\mathscr{M}} \text{ skončí výpočet nad } K(\hat{\mathscr{M}}) \Leftrightarrow K(\hat{\mathscr{M}}) \in L_{\text{neg}}.$$

Podle definice jazyka L_{neg}, ale

$$K(\hat{\mathscr{M}}) \in L_{\text{neg}} \Leftrightarrow \hat{\mathscr{M}} \text{ neskončí výpočet nad } K(\hat{\mathscr{M}}).$$

Tím jsme dospěli ke sporu. Jazyk L_U tedy není rekurzívní.

Skutečnost, že L_U není rekurzívní, bývá zvykem stručně vyjadřovat touto formulací:

Fakt 6.2.7. Problém zastavení Turingova stroje je algoritmicky nerozhodnutelný.

Tento fakt slouží jako východisko pro dokazování algoritmické nerozhodnutelnosti dalších problémů, na něž je problém zastavení převeditelný. Jestliže se totiž podaří dokázat, že z algoritmické rozhodnutelnosti nějakého problému P by logicky vyplývala algoritmická rozhodnutelnost problému zastavení Turingova stroje, je podle 6.2.7 i problém P algoritmicky nerozhodnutelný. V následující větě ukážeme velmi jednoduchý příklad převedení jistého problému na problém zastavení Turingova stroje.

Věta 6.2.8. *Neexistuje algoritmus, který by pro libovolný Turingův stroj $\mathscr{M} \in \mathbf{M}$ a slovo $w \in \{0, 1\}^*$ rozhodoval, zda w je přijímáno strojem $\mathscr{M}$.*

Důkaz. Existuje algoritmus $\mathscr{A}$, který ke každému Turingovu stroji $\mathscr{M} \in \mathbf{M}$ zadanému kódem $K(\mathscr{M})$ sestrojí kód $K(\hat{\mathscr{M}})$ stroje $\hat{\mathscr{M}} \in \mathbf{M}$ definovaného takto: jestliže

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

potom

$$\hat{\mathcal{M}} = (Q \cup \{q_f\}, \Sigma, \delta', q_0, \{q_f\}),$$

kde q_f je nově přidaný stav nepatřící do Q a $\delta'(q, x) = (q_f, x, +1)$, jakmile $\delta(q, x)$ není definována. V ostatních případech se δ' rovná δ. Zřejmě platí, že pro libovolné slovo $w \in \{0, 1\}^*$ $\hat{\mathcal{M}}$ přijímá w, právě když $\mathcal{M}$ skončí výpočet nad w.

Kdyby tedy existoval algoritmus $\mathcal{B}$ rozhodující pro libovolný stroj $\mathcal{M}$ a w, zda $\mathcal{M}$ přijímá w, potom by bylo možné algoritmicky rozhodovat problém zastavení Turingova stroje takto: pro zadané $\mathcal{M} \in \mathbf{M}$ a $w \in \{0, 1\}^*$ by se pomocí algoritmu $\mathcal{A}$ sestrojil stroj $\hat{\mathcal{M}}$ a pomocí $\mathcal{B}$ by se rozhodlo, zda $\hat{\mathcal{M}}$ přijímá w. Tím by bylo zjištěno, zda $\mathcal{M}$ se nad w zastaví, neboť $\mathcal{M}$ se zastaví nad $w \Leftrightarrow \hat{\mathcal{M}}$ přijímá w.

Obdobným způsobem se dokazuje algoritmická nerozhodnutelnost tzv. Postova korespondenčního problému, s nímž se nyní seznámíme a kterého se s výhodou využívá při důkazu nerozhodnutelnosti některých problémů z teorie formálních jazyků.

Definice 6.2.9. *Postovým korespondenčním problémem* (PKP) budeme nazývat každou dvojici

$$\langle A, B \rangle,$$

kde

$$A = u_1, u_2, \ldots, u_n, \quad B = v_1, v_2, \ldots, v_n$$

pro nějaké $n \geqq 1$ a

$$u_1, \ldots, u_n, v_1, \ldots, v_n \in \Sigma^+$$

pro nějakou abecedu Σ.

Řekneme, že PKP $\langle A, B \rangle$ *má řešení*, jestliže existují indexy $i_1, \ldots, i_r$ tak, že

$$1 \leqq i_1, \ldots, i_r \leqq n$$

a

$$u_{i_1} u_{i_2} \ldots u_{i_r} = v_{i_1} v_{i_2} \ldots v_{i_r}.$$

Příklad 6.2.10. PKP $\langle A, B\rangle$ zadaný tabulkou

	A	B
1	1	10
2	1	1100
3	001	0
4	0	110

má např. řešení 331324:

$$u_3u_3u_1u_3u_2u_4 = 001001100110 = v_3v_3v_1v_3v_2v_4 .$$

Avšak např. PKP $\langle C, D\rangle$ zadaný tabulkou

	C	D
1	001	0
2	1	001

řešení nemá. Kdyby totiž řešení měl, začínalo by nutně první dvojicí: 001, protože druhá dvojice se liší v prvním symbolu.
0
Po tomto vynuceném začátku není možné pokračovat druhou dvojicí: 0011, protože u třetího symbolu začíná neshoda. Zbývá
0001
jen pokračování první dvojicí: 001001, které už nelze dále pro-
00
dloužit, protože obě slova v seznamu D začínají symbolem 0.

Pro dva konkrétní problémy se nám v předchozím příkladu podařilo rozhodnout, zda mají řešení. Jednoduchost formulace úlohy může svádět k domněnce, že existuje metoda, jak pro každý PKP rozhodnout, zda má řešení. Taková domněnka je však nesprávná.

Věta 6.2.11. *Neexistuje algoritmus, který by pro každý* PKP *rozhodoval, zda má řešení.*

Podrobný důkaz této věty je poměrně dlouhý a lze jej nalézt např v [20]. Spočívá v převedení problému zastavení Turingova stroje na problém řešitelnosti PKP. Je totiž možné sestrojit algoritmus $\mathscr{A}$, který na základě kódu libovolného Turingova stroje $\mathscr{M}$ (viz důkaz 6.2.4) a vstupního slova w sestrojí PKP $\langle A_{\mathscr{M},w}, B_{\mathscr{M},w}\rangle$, který má řešení právě tehdy, jestliže $\mathscr{M}$ provádí nad w konečný výpočet. Kdyby tedy existoval algoritmus $\mathscr{B}$ rozhodující řešitelnost PKP, bylo by možné rozhodovat problém zastavení Turingova stroje takto: $\mathscr{M}$ se zastaví nad w, právě když algoritmus $\mathscr{B}$ určí, že PKP $\langle A_{\mathscr{M},w}, B_{\mathscr{M},w}\rangle$ získaný pomocí algoritmu $\mathscr{A}$ má řešení. Proto podle 6.2.7 algoritmus $\mathscr{B}$ nemůže existovat.

Význam věty tkví v tom, že problém řešitelnosti PKP se poměrně snadno převádí na řadu problémů z teorie formálních jazyků, čímž se dokazuje jejich algoritmická nerozhodnutelnost. Příkladem je následující věta.

Věta 6.2.12. *Neexistuje algoritmus, který by pro libovolné dvě bezkontextové gramatiky $\mathscr{G}_1, \mathscr{G}_2$ rozhodoval, zda*

$$L(\mathscr{G}_1) \cap L(\mathscr{G}_2) = \emptyset .$$

Důkaz. Předpokládejme, že takový algoritmus existuje. Stačí ukázat, že bychom tohoto algoritmu mohli použít k rozhodování řešitelnosti PKP.

Buď $\langle A, B\rangle$ libovolný PKP, $A = u_1, \ldots, u_n$; $B = v_1, \ldots, v_n$. Zvolme libovolně nově přidané symboly $a_1, \ldots, a_n$, které se nevyskytují v žádném u_i ani v_i. Dále sestrojme bezkontextové gramatiky

$$\mathscr{G}_A\colon\ S \to u_i S a_i \mid u_i a_i ,\ 1 \leqq i \leqq n ,$$
$$\mathscr{G}_B\colon\ S \to v_i S a_i \mid v_i a_i ,\ 1 \leqq i \leqq n .$$

Potom $\langle A, B\rangle$ má řešení, právě když $L(\mathscr{G}_A) \cap L(\mathscr{G}_B) \neq \emptyset$. Jestliže totiž

$$u_{i_1} \ldots u_{i_r} a_{i_r} \ldots a_{i_1} = v_{j_1} \ldots v_{j_s} a_{j_s} \ldots a_{j_1} \in L(\mathscr{G}_A) \cap L(\mathscr{G}_B)$$

pro nějaká $i_1, \ldots, i_r$ a $j_1, \ldots, j_s$, musí nutně být $r = s$ a $i_m = j_m$ pro všechna m, $1 \leqq m \leqq r$. To plyne z toho, že se v obou částech

musí rovnat druhé části slov tvořené symboly a_i. Jelikož se ale musí rovnat i první části slov, je nutně

$$u_{i_1} \dots u_{i_r} = v_{i_1} \dots v_{i_r}$$

a $\langle A, B \rangle$ má řešení.

Obráceně, jestliže $\langle A, B \rangle$ má řešení $i_1, \dots, i_k$, je

$$u_{i_1} \dots u_{i_k} a_{i_k} \dots a_{i_1} = v_{i_1} \dots v_{i_k} a_{i_k} \dots a_{i_1} \in L(\mathscr{G}_A) \cap L(\mathscr{G}_B).$$

Podobně se řešitelnost PKP převádí i na řešitelnost dalších problémů. Následující věta je výčtem některých z nich. Důkaz všech tvrzení této věty je možné najít např. v knize [20].

Věta 6.2.13. *Následující problémy jsou algoritmicky nerozhodnutelné*:

1. *Pro libovolnou bezkontextovou gramatiku* $\mathscr{G} = (\Pi, \Sigma, S, P)$ *rozhodnout, zda* $L(\mathscr{G}) = \Sigma^*$.
2. *Pro libovolnou bezkontextovou gramatiku* $\mathscr{G}_1$ *a regulární gramatiku* $\mathscr{G}_2$ *rozhodnout, zda* $L(\mathscr{G}_1) = L(\mathscr{G}_2)$, *resp.*
3. *rozhodnout, zda* $L(\mathscr{G}_1) \supseteq L(\mathscr{G}_2)$.
4. *Rozhodnout pro libovolné bezkontextové gramatiky* $\mathscr{G}_1$, $\mathscr{G}_2$, *zda* $L(\mathscr{G}_1) = L(\mathscr{G}_2)$, *resp.*
5. *rozhodnout, zda* $L(\mathscr{G}_1) \supseteq L(\mathscr{G}_2)$, *resp.*
6. *rozhodnout, zda* $L(\mathscr{G}_1) \cap L(\mathscr{G}_2)$ *je bezkontextový jazyk.*
7. *Rozhodnout pro libovolnou bezkontextovou gramatiku, zda* $\overline{L(\mathscr{G})}$ *je bezkontextový jazyk, resp.*
8. *rozhodnout, zda* $L(\mathscr{G})$ *je regulární jazyk, resp.*
9. *rozhodnout, zda gramatika* $\mathscr{G}$ *je víceznačná.*
10. *Rozhodnout pro libovolné dva deterministické zásobníkové automaty* $\mathscr{M}_1$, $\mathscr{M}_2$, *zda* $L(\mathscr{M}_1) \cap L(\mathscr{M}_2) = \emptyset$, *resp.*
11. *rozhodnout, zda* $L(\mathscr{M}_1) \cap L(\mathscr{M}_2)$ *je bezkontextový jazyk, resp.*
12. *rozhodnout, zda* $L(\mathscr{M}_1) \cup L(\mathscr{M}_2)$ *je deterministický, resp.*
13. *rozhodnout, zda* $L(\mathscr{M}_1) \subseteq L(\mathscr{M}_2)$.
14. *Rozhodnout pro libovolnou kontextovou gramatiku, zda* $L(\mathscr{G}) = \emptyset$.

Poznámka 6.2.14. Je zajímavé srovnat tvrzení 1 a 14 předchozí věty s větou 4.1.4, podle které lze pro každou bezkontextovou

gramatiku rozhodnout, zda generuje prázdný jazyk. Podobně tvrzení 2 a 4 kontrastují s př. 2.2.14, podle kterého lze rozhodnout rovnost regulárních jazyků.

Za zmínku také stojí, že i když podle tvrzení 3 nelze pro libovolný bezkontextový jazyk L a libovolný regulární jazyk R rozhodovat, zda $L \supseteq R$, je opačná inkluze $L \subseteq R$ snadno rozhodnutelná. Je totiž $L \subseteq R \Leftrightarrow L \cap \bar{R} = \emptyset$ a $L \cap \bar{R}$ je podle 4.5.10 bezkontextový jazyk, a tedy lze rozhodnout, zda je prázdný.

ROZŠIŘUJÍCÍ ČÁST

6.3. Nerozhodnutelnost příslušnosti k LR a LL gramatikám

LR a LL jsou podle výsledků kap. 5 důležitými podtřídami bezkontextových gramatik umožňujícími deterministickou syntaktickou analýzu zdola, resp. shora. Podle výsledků čl. 5.4 a 5.6 je možné pro libovolnou bezkontextovou gramatiku $\mathscr{G}$ a přirozené číslo k algoritmicky rozhodnout, zda je LR(k), resp. LL(k). Obecnější otázka je, zda daná gramatika je LR(k), resp. LL(k) vůbec pro nějaké k. K odpovědi na tuto otázku zřejmě algoritmy z čl. 5.4 a 5.6 nestačí. Pomocí nich můžeme např. testovat, zda je daná gramatika LR(k) postupně pro $k = 0, 1, 2, \ldots$ atd. Jestliže gramatika je skutečně LR, dospějeme po konečném počtu kroků k výsledku. Pokud však gramatika není LR, bude testovací proces probíhat do nekonečna. Výsledky kap. 5 tedy nechávají otázku, zda příslušnost k LR, resp. LL gramatikám je algoritmicky rozhodnutelná, otevřenou. V tomto článku zmíněnou otázku zodpovíme záporně.

Nejprve zavedeme dva pomocné pojmy.

Definice 6.3.1. *Korespondenčním systémem* nazveme každou dvojici $\langle A, B \rangle$, kde $A = u_1, \ldots, u_n$; $B = v_1, \ldots, v_n$ pro nějaké $n \geq 1$ a neprázdná slova $u_1, \ldots, u_n, v_1, \ldots, v_n$.

Jestliže u, v jsou libovolná dvě slova, z nichž jedno je prefixem (počátečním úsekem) druhého, a jestliže existují indexy $i_1, \ldots, i_k$ takové, že jedno ze slov

$$x = uu_{i_1} \ldots u_{i_k}, \quad y = vv_{i_1} \ldots v_{i_k}$$

je prefixem druhého, potom řekneme, že $\langle A, B\rangle$ *prodlužuje dvojici* (u, v) *na* (x, y).

To znamená například, že PKP se zadává korespondenčním systémem a má řešení, právě když příslušný systém prodlužuje dvojici (e, e) na nějakou dvojici (u, u), kde u je neprázdné.

Pro všechny další výsledky v tomto článku i např. pro důkaz nerozhodnutelnosti PKP má rozhodující význam mechanismus, kterým lze pomocí korespondenčních systémů a prodlužování dvojic slov simulovat výpočet Turingova stroje. Tento mechanismus nyní popíšeme.

Definice 6.3.2. Nechť $\mathcal{M} = (Q, \Sigma, \delta, q_0, F)$ je libovolný Turingův stroj a $\#$, $\bullet$ dva nově přidané symboly. Definujme *korespondenční systém* $\mathcal{S}_1(\mathcal{M}) = \langle C_{\mathcal{M}}, D_{\mathcal{M}}\rangle$ takto:

	$C_{\mathcal{M}}$	$D_{\mathcal{M}}$	
I.	$\bullet\, a$	$a\,\bullet$	pro všechna $a \in \Sigma \cup \{\#\}$,
II.	$\bullet\, q\,\bullet\, a$	$a'\,\bullet\, q'\,\bullet$	jestliže $\delta(q, a) = (q', a', +1)$,
III.	$\bullet\, q\,\bullet\, \#$	$a'\,\bullet\, q'\,\bullet\, \#\,\bullet$	jestliže $\delta(q, b) = (q', a', +1)$,
IV.	$\bullet\, c\,\bullet\, q\,\bullet\, a$	$q'\,\bullet\, c\,\bullet\, a'\,\bullet$	jestliže $\delta(q, a) = (q', a', -1)$,
V.	$\bullet\, c\,\bullet\, q\,\bullet\, \#$	$q'\,\bullet\, c\,\bullet\, a'\,\bullet\, \#\,\bullet$	jestliže $\delta(q, b) = (q', a', -1)$,
VI.	$\bullet\, \#\,\bullet\, q\,\bullet\, a$	$\#\,\bullet\, q'\,\bullet\, b\,\bullet\, a'\,\bullet$	jestliže $\delta(q, a) = (q', a', -1)$.

Řádky v tabulce reprezentují vždy celý systém dvojic daný podmínkami vpravo, v nichž vždy $q \in Q$, $a, c \in \Sigma$ a $b \in \Sigma$ označuje prázdný symbol.

Pro několik dalších odstavců se domluvme, že pro libovolný řetěz $w = d_1 d_2 \ldots d_n$ (kde d_i jsou symboly abecedy) budeme řetěz $d_1 \bullet d_2 \bullet \ldots \bullet d_n$ označovat symbolem $\tilde{w}$.

Lemma 6.3.3. *Nechť K_1 je libovolná konfigurace Turingova stroje $\mathscr{M}$ a u nějaké slovo. Potom platí tato dvě tvrzení:*

1. *Jestliže K_2 je konfigurace stroje $\mathscr{M}$ taková, že $K_1 \vdash_{\mathscr{M}} K_2$, pak korespondenční systém $\mathscr{S}_1(\mathscr{M})$ prodlužuje dvojici*

$$(u\,\#, \widetilde{u\,\#\,K_1\,\#\,\bullet})$$

na dvojici

$$(\widetilde{u\,\#\,K_1\,\#}, \widetilde{u\,\#\,K_1\,\#\,K_2\,\#\,\bullet}).$$

2. *Jestliže $\mathscr{S}_1(\mathscr{M})$ prodlužuje dvojici*

$$(u\,\#, \widetilde{u\,\#\,K_1\,\#\,\bullet})$$

na nějakou dvojici

$$(u\,\#\,z_1, \widetilde{u\,\#\,K_1\,\#\,\bullet}\,z_2)$$

takovou, že

$$|z_1| \geq |\widetilde{\bullet\,K_1\,\#}|,$$

potom $\bullet\,K_1\,\#$ je počátečním úsekem z_1, existuje (jediná) konfigurace K_2 taková, že $K_1 \vdash_{\mathscr{M}} K_2$, $\widetilde{K_2\,\#\,\bullet}$ je počátečním úsekem z_2 a $(\widetilde{u\,\#\,K_1\,\#}, \widetilde{u\,\#\,K_1\,\#\,K_2\,\#\,\bullet})$ je prodloužením

$$(u\,\#, \widetilde{u\,\#\,K_1\,\#\,\bullet}).$$

Pro větší přehlednost budeme složky dvojic zapisovat pod sebe.

Důkaz. 1. Budiž $K_1 = x_1 cqax_2$ pro nějaká $x_1, x_2 \in \Sigma^*$, $a, c \in \Sigma \cup \{e\}$ a $q \in Q$ a nechť např. $\delta(q, a) = (q', a', -1)$. Potom použitím dvojic typu I z $\mathscr{S}_1(\mathscr{M})$ (viz 6.3.2) je možné dvojici

$$u\,\#\,,$$

$$u\,\#\,\bullet\,\tilde{x}_1\,\bullet\,c\,\bullet\,q\,\bullet\,a\,\bullet\,\tilde{x}_2\,\bullet\,\#\,\bullet$$

prodloužit na

$$u \# \bullet \tilde{x}_1 ,$$

$$u \# \bullet \tilde{x}_1 \bullet c \bullet q \bullet a \bullet \tilde{x}_2 \bullet \# \bullet \tilde{x}_1 \bullet .$$

Použitím vhodné dvojice typu IV (resp. V, pokud $a = x_2 = e$, resp. VI, pokud $x_1 = c = e$) tuto dvojici dále prodloužíme na

$$u \# \bullet \tilde{x}_1 \bullet c \bullet q \bullet a ,$$

$$u \# \bullet \tilde{x}_1 \bullet c \bullet q \bullet a \bullet \tilde{x}_2 \bullet \# \bullet \tilde{x}_1 \bullet q' \bullet c \bullet a' \bullet$$

a opětovným použitím dvojic typu I dostaneme definitivně

$$\widetilde{u \# K_1 \#} ,$$

$$\widetilde{u \# K_1 \# K_2 \# \bullet} ,$$

kde

$$K_2 = x_1 q' c a' x_2 .$$

Analogicky při kroku s pohybem hlavy doprava.

2. Má-li se první slovo prodloužit o délku $|\widetilde{\bullet K_1 \#}|$, musí to být o odpovídající úsek druhého slova, tj. o $\widetilde{\bullet K_1 \#}$. Jestliže $K_1 = x_1 cqax_2$, potom z tvaru $\mathscr{S}_1(\mathscr{M})$ je zřejmé, že dvojici

$$(u \#,\ u \# \bullet \tilde{x}_1 \bullet c \bullet q \bullet a \bullet \tilde{x}_2 \bullet \# \bullet)$$

lze v první fázi (odpovídající $\tilde{x}_1$, popřípadě $\widetilde{x_1 c}$) prodlužovat pouze dvojicemi typu I. Prodloužit první slovo o úsek odpovídající q lze pouze za použití dvojice typu II až VI. Příslušnou dvojici lze nalézt, pouze když existuje konfigurace K_2 taková, že $K_1 \vdash_{\mathscr{M}} K_2$. Její volba je pak jednoznačná díky tomu, že $\mathscr{M}$ je deterministický stroj.

Následným doplněním prvního slova dvojicemi typu I na $\widetilde{u \# K_1 \#}$ se ve druhém slově doplní zbytek řetězu $\widetilde{K_2 \# \bullet}$.

Poznámka 6.3.4. Z předchozího lemmatu ještě není zřejmý význam prokládání slov symbolem $\bullet$. Lemma by platilo, i kdybychom v něm a v definici 6.3.2 vypustili všechny výskyty $\bullet$.

Smysl zavedení tohoto symbolu vysvitne až z několika následujících důkazů.

Definice 6.3.5. Nechť $\mathscr{M}$ je libovolný Turingův stroj a w libovolné slovo. Označme

$$C_{\mathscr{M}} = u_1, \ldots, u_n; \; D_{\mathscr{M}} = v_1, \ldots, v_n,$$

kde

$$\langle C, D \rangle = \mathscr{S}_1(\mathscr{M})$$

(viz 6.1.2). Potom *korespondenční systém*

$$\mathscr{S}_1(\mathscr{M}, w) = \langle C_{\mathscr{M},w}, D_{\mathscr{M},w} \rangle$$

definujeme takto:

	$C_{\mathscr{M},w}$	$D_{\mathscr{M},w}$
1	$\#$	$\# \, bq_0w \, \# \bullet$
2	u_1	v_1
3	u_2	v_2
$\vdots$	$\vdots$	$\vdots$
$n+1$	u_n	v_n

Lemma 6.3.6. *Nechť $\mathscr{M}$ je libovolný Turingův stroj a w libovolný řetěz. Potom tyto dvě podmínky jsou ekvivalentní:*

1. *Výpočet $\mathscr{M}$ nad w je nekonečný.*
2. *Ke každému $n \geqq 1$ existují slova u, v taková, že*

$$n \leqq \min(|u|, |v|)$$

a (u, v) je prodloužením dvojice (e, e) pomocí korespondenčního systému $\mathscr{S}_1(\mathscr{M}, w)$.

Důkaz. $1 \Rightarrow 2$: Jestliže existuje nekonečná posloupnost konfigurací

$$bq_0w = K_1 \vdash_{\mathscr{M}} K_2 \vdash_{\mathscr{M}} K_3 \vdash_{\mathscr{M}} \ldots,$$

potom dvojici (e, e) lze pomocí první dvojice z $\mathscr{S}_1(\mathscr{M}, w)$ pro-

dloužit na dvojici $(\#, \widetilde{\# K_1 \# \bullet})$ a tuto dvojici je možné podle lemmatu 6.3.3 neomezeně dále prodlužovat.

$2 \Rightarrow 1$: Každé prodloužení dvojice (e, e) musí začínat první dvojicí systému $\mathscr{S}_1(\mathscr{M}, w)$, protože všechny ostatní dvojice se liší prvními symboly (poprvé využíváme $\bullet$!). Dále už se tato dvojice nikdy neuplatní. Vzniklá dvojice a každé její prodloužení pomocí dvojic 2 až $n + 1$ se totiž vyznačují tím, že druhé slovo dvojice končí na $\bullet$, zatímco první končí na symbol různý od $\bullet$ a v takové situaci nelze dvojici 1 použít.

Každé prodloužení (e, e) je tedy prodloužením

$$(\#, \widetilde{\# bq_0w \# \bullet})$$

pomocí dvojic 2 až $n + 1$, tzn. pomocí systému $\mathscr{S}_1(\mathscr{M})$. Jestliže existuje libovolně dlouhé takové prodloužení, potom podle 6.3.2 existuje nekonečná posloupnost konfigurací

$$bq_0w \vdash_{\mathscr{M}} K_2 \vdash_{\mathscr{M}} \dots,$$

tzn. výpočet $\mathscr{M}$ nad w neskončí.

Věta 6.3.7. *Neexistuje algoritmus, který by pro každou bezkontextovou gramatiku rozhodoval, zda je to* LL *gramatika.*

Důkaz. Převedeme problém zastavení Turingova stroje na problém příslušnosti k LL gramatikám. Tím bude podle 6.2.7 věta dokázána. K libovolnému Turingovu stroji $\mathscr{M}$ a slovu w sestrojme podle 6.3.5 korespondenční systém

$$\mathscr{S}_1(\mathscr{M}, w) = \langle C_{\mathscr{M},w}, D_{\mathscr{M},w} \rangle .$$

Označme

$$C_{\mathscr{M},w} = x_1, \dots, x_n, \; D_{\mathscr{M},w} = y_1, \dots, y_n .$$

Buďte $a_1, \dots, a_n$ nově přidané, vzájemně různé symboly. Sestrojme gramatiku $\mathscr{G}_{\mathscr{M},w}$:

$$S \to a_i S A_i \quad \text{pro všechna } i,\ 2 \leqq i \leqq n,$$

$$S \to a_1 A_1 ,$$

$$\left.\begin{matrix} A_i \to x_i \\ A_i \to y_i \end{matrix}\right\} \quad \text{pro všechna } i,\ 1 \leqq i \leqq n.$$

$S, A_1, \ldots, A_n$ jsou neterminály, S je počáteční symbol.

Ukážeme, že tato gramatika je LL, právě když pro nějaké n neexistuje žádné prodloužení (u, v) dvojice (e, e) systémem

$$\mathscr{S}_1(\mathscr{M}, w),$$

které by splňovalo podmínku $n < \min(|u|, |v|)$. To podle 6.3.6 bude znamenat, že $\mathscr{G}_{\mathscr{M},w}$ je LL, právě když výpočet $\mathscr{M}$ nad w je konečný. Problém zastavení Turingova stroje tak bude převeden na problém příslušnosti k LL gramatikám.

Všechna S-pravidla gramatiky $\mathscr{G}_{\mathscr{M},w}$ se liší již prvním symbolem, který je ve všech případech terminální. Rovněž A_i-pravidla pro všechna $i \geqq 2$ liší již v prvních symbolech, které jsou opět terminální, neboť podle 6.3.2 a 6.3.5 pravidla $A_i \to x_i$ (pro $i \geqq 2$) začínají symbolem ●, zatímco pravidla $A_i \to y_i$ $(i \geqq 2)$ začínají symbolem různým od ●.

Všechna S-pravidla a všechna A_i-pravidla pro $i \geqq 2$ tedy vyhovují už podmínkám pro LL(1) gramatiky. Z toho plyne, že se stačí soustředit na dvojici pravidel

$$A_1 \to x_1,$$

$$A_1 \to y_1.$$

Všechny větné formy, ve kterých se při levé derivaci objeví A_1, jsou právě řetězy tvaru

$$(*) \qquad a_{i_1} \ldots a_{i_2} a_1 A_1 A_{i_2} \ldots A_{i_r}$$

pro libovolná $2 \leqq i_2, \ldots, i_r \leqq n$, $r \geqq 0$.

Takový řetěz lze dále přepsat na terminální slovo tvaru

$$(**) \qquad a_{i_1} \ldots a_{i_2} a_1 w_1 w_{i_2} \ldots w_{i_r},$$

kde pro každé $j \geqq 1$ je $w_{i_j} = x_{i_j}$ nebo $w_{i_j} = y_{i_j}$.

Zde je třeba upozornit na důležitou vlastnost slov x_i a y_i (viz 6.3.2). Jestliže se totiž v $(*)$ přepíše nějaký neterminál A_i pra-

vidlem typu $A_i \to x_i$ a bezprostředně následující neterminál A_j pravidlem typu $A_j \to y_j$, projeví se to na příslušném místě slova (**) výskytem dvou sousedících symbolů různých od ●. Jestliže naopak dvojici A_iA_j přepíšeme pravidly typu $A_i \to y_i$ a $A_j \to x_j$, projeví se to ve slově (**) výskytem dvojice ●●. (To je další důvod, proč jsme zavedli ● – viz poznámku 6.3.4.)

Gramatika $\mathscr{G}_{\mathscr{M},w}$ není LL, jestliže pro každé k lze nalézt indexy $i_2, \ldots, i_r$ takové, že

$$k(x_1 w_{i_2}^1 \ldots w_{i_r}^1) = k(y_1 w_{i_2}^2 \ldots w_{i_r}^2),$$

kde každé $w_{i_j}^1$ a $w_{i_j}^2$ je rovno buď x_{i_j}, nebo y_{i_j}. Na základě předchozí úvahy se stačí omezit na případ

$$(***) \qquad k(x_1 x_{i_2} \ldots x_{i_r}) = k(y_1 y_{i_2} \ldots y_{i_r}).$$

V ostatních případech je možné z prvního výskytu dvojice ●● (resp. dvojice symbolů různých od ●) usoudit, že ve slově je $w_1 = y_1$ (resp. $w_1 = x_1$).

Pokud pro každé k existují $i_2, \ldots, i_r$ tak, že platí (***), znamená to, že systémem $\mathscr{S}_1(\mathscr{M}, w)$ lze dvojici (e, e) libovolně prodloužit, tzn. (podle 6.3.6) výpočet stroje $\mathscr{M}$ nad slovem w je nekonečný.

Souhrnně řečeno, $\mathscr{G}_{\mathscr{M},w}$ není LL, právě když $\mathscr{M}$ se nad w nezastaví. Tím je důkaz dokončen.

Nerozhodnutelnost příslušnosti k LR gramatikám dokážeme pomocí výsledku o nerozhodnutelnosti tzv. problému částečné korespondence.

Definice 6.3.8. *Problém částečné korespondence* (PČK) zadáváme libovolným korespondenčním systémem $\mathscr{S} = \langle C, D \rangle$. Řekneme, že PČK $\langle C, D \rangle$ *má řešení*, jestliže dvojici (e, e) lze systémem $\mathscr{S}$ libovolně prodloužit, tzn. jestliže ke každému číslu $n \geqq 0$ existuje prodloužení (u, v) dvojice (e, e) pomocí $\mathscr{S}$ takové, že

$$n < \min(|u|, |v|).$$

Příklad 6.3.9. PČK $\langle A, B \rangle$ zadaný tabulkou

A	B
01	011
10	01

má zřejmě řešení, neboť pro libovolné číslo $p > 1$ stačí vzít posloupnost indexů $1, 2, 2, \ldots, 2$ délky p, abychom dostali prodloužení

$$u_1 u_2 u_2 \ldots u_2 = 011010 \ldots 10,$$
$$v_1 v_2 v_2 \ldots v_2 = 0110101 \ldots 01.$$

Příklad 6.3.10. PČK $\langle A, B \rangle$ zadaný tabulkou

	A	B
1	*bba*	*bb*
2	*bab*	*abaa*
3	*ab*	*abb*

řešení nemá, jak můžeme zjistit rozborem těchto případů:

1. Každá posloupnost indexů začínající indexem 2 vede na dvojici slov

$$bab \ldots,$$
$$abaa \ldots,$$

která se liší už v prvním symbolu.

2. Posloupnosti indexů začínající $1, 1, \ldots$ vedou na slovo lišící se na 5. místě, posloupnosti $1, 2, \ldots$ vedou k neshodě na 6. místě, posloupnosti $1, 3, \ldots$ vedou k neshodě na 4. místě.

3. U posloupností začínajících $3, 1, \ldots$ nastane neshoda v 5. symbolu, posloupnosti $3, 2, \ldots$ vedou k neshodě nejpozději na 7. místě (u posloupností začínajících $3, 2, 3, \ldots$), u posloupností $3, 3, \ldots$ nastane neshoda na 3. místě.

Věta 6.3.11. *Neexistuje algoritmus, který by pro každý* PČK *rozhodoval, zda má řešení.*

Důkaz. Pro libovolný Turingův stroj $\mathscr{M}$ a slovo w má PČK zadaný systémem

$$\mathscr{S}_1(\mathscr{M}, w) = \langle C_{\mathscr{M},w}, D_{\mathscr{M},w} \rangle$$

(viz 6.3.5) řešení, právě když výpočet $\mathscr{M}$ nad w je nekonečný (podle 6.3.6). Problém řešitelnosti PČK je proto podle 6.2.7 nerozhodnutelný.

Věta 6.3.12. *Neexistuje algoritmus, který by pro každou bezkontextovou gramatiku rozhodoval, zda je to* LR *gramatika.*

Důkaz. Převedeme PČK na problém příslušnosti k LR gramatikám, čímž bude podle 6.3.11 věta dokázána.

Budiž $\langle A, B \rangle$, kde $A = u_1, \ldots, u_n$ a $B = v_1, \ldots, v_n$, kde $u_i, v_i \in \Sigma^+$, libovolný PČK. Zvolme n nových vzájemně různých symbolů $a_1, \ldots, a_n$ a sestrojme bezkontextovou gramatiku $\mathscr{G}_{AB}$:

$$S \to A \mid B,$$

$$A \to a_i A u_i \mid a_i u_i \quad \text{pro každé } i,\ 1 \leqq i \leqq n,$$

$$B \to a_i B v_i \mid a_i v_i \quad \text{pro každé } i,\ 1 \leqq i \leqq n.$$

Ukážeme, že $\mathscr{G}_{AB}$ je LR, právě když PČK $\langle A, B \rangle$ nemá řešení. Zvolme libovolné $k \geqq 1$ a uvažujme, jak vypadá k-položkový automat pro $\mathscr{G}_{AB}$. Zjistíme, v kterém případě dochází v nějakém stavu automatu ke kolizi mezi dvěma položkami, působící, že $\mathscr{G}_{AB}$ není LR(k).

Počáteční stav tohoto automatu je tvořen k-položkami

$$\begin{aligned}
&S \to .A, e,\\
&S \to .B, e,\\
&\left.\begin{aligned}
&A \to .a_i A u_i, e\\
&A \to .a_i u_i, e\\
&B \to .a_i B v_i, e\\
&B \to .a_i v_i, e
\end{aligned}\right\} \text{pro všechna } i,\ 1 \leqq i \leqq n.
\end{aligned}$$

V tohoto stavu lze přejít jednak vstupy A, resp. B do stavů $\{N \to A., e\}$, resp. $\{S \to B., e\}$, které jsou bez kolizí, jednak na základě slov typu $a_{i_1} \dots a_{i_r}$ $(r \leqq k)$ do stavů tvořených k-položkami

$$(*) \quad \left\{ \begin{array}{l} A \to a_{i_r} .A u_{i_r}, k(u_{i_{r-1}} \dots u_{i_1}), \\ A \to a_{i_r} .u_{i_r}, k(u_{i_{r-1}} \dots u_{i_1}), \\ B \to a_{i_r} .B v_{i_r}, k(v_{i_{r-1}} \dots v_{i_1}), \\ B \to a_{i_r} .v_{i_r}, k(v_{i_{r-1}} \dots v_{i_1}), \\ A \to .a_j A u_j, k(u_{i_r} \dots u_{i_1}) \\ A \to .a_j u_j, k(u_{i_r} \dots u_{i_1}) \\ B \to .a_j B v_j, k(v_{i_r} \dots v_{i_1}) \\ B \to .a_j v_j, k(v_{i_r} \dots v_{i_1}) \end{array} \right\} \text{pro všechna } j, \; 1 \leqq j \leqq n .$$

Tyto stavy ještě neobsahují žádnou úplnou k-položku a nedochází v nich ke kolizím. Z každého stavu typu $(*)$ se na základě symbolů a_i přechází opět pouze do stavů typu $(*)$, poněvadž libovolné $u_{i_k} \dots u_{i_1}$, resp. $v_{i_k} \dots v_{i_1}$ už má délku alespoň k. Na základě A se ze stavů $(*)$ přechází do stavů typu $\{A \to a_i A .u_i, \dots\}$. Každý z těchto stavů i všechny stavy z nich dosažitelné obsahují nejvýše jednu položku, a nevzniká v nich proto kolize. Podobně je tomu u přechodu z $(*)$ pomocí symbolu B. Kolize tedy mohou vznikat nejvýše ve stavech, do nichž se lze ze stavů typu $(*)$ dostat na základě řetězů symbolů z abecedy Σ. Tyto stavy jsou buď prázdné množiny, nebo jsou tvořeny dvojicemi k-položek tvaru

$$(**) \quad \left\{ \begin{array}{l} A \to a_j u_j^1 .u_j^2, k(u_{i_r} \dots u_{i_1}) \\ B \to a_j v_j^1 .v_j^2, k(u_{i_r} \dots u_{i_1}) \end{array} \right.$$

pro nějaké $r \leqq k$, $1 \leqq i_1, \dots, i_r \leqq n$, kde

$$u_j^1 u_j^2 = u_j, \; v_j^1 v_j^2 = v_j \text{ a } u_j^1 = v_j^1 .$$

Položka typu $(**)$ obsahuje kolizi (viz 5.4.7), právě když

$$(1) \qquad u_j^2 = e \text{ a } k(u_{i_r} \dots u_{i_1}) = k(v_j^2 v_{i_r} \dots v_{i_1})$$

nebo

$$(2) \qquad v_j^2 = e \text{ a } k(v_{i_r} \dots v_{i_1}) = k(u_j^2 u_{i_r} \dots u_{i_1}) .$$

To ovšem nastane právě tehdy, když pro

$$s = k + |u_j^1| = k + \min(|u_j|, |v_j|)$$

je

$$(***) \qquad s(u_j u_{i_r} \dots u_{i_1}) = s(v_j v_{i_r} \dots v_{i_1}).$$

Z toho je vidět, že pokud $\langle A, B\rangle$ nemá řešení, budou pro dostatečně velké k všechny stavy k-položkového automatu bez kolize a $\mathscr{G}_{AB}$ je LR(k). Obráceně, jestliže $\langle A, B\rangle$ má řešení, najde se pro každé k taková $(r+1)$-tice indexů $(r \leqq k)$ $i_1, \dots, i_r, j$, že bude splněna podmínka (***), tzn. příslušný stav (**) obsahuje kolizi, tj. $\mathscr{G}_{AB}$ není LR(k) pro žádné k.

6.4. Výpočtová složitost

Na otázku, zda určitý problém je algoritmicky řešitelný, je možné navázat další otázkou: jestliže už problém algoritmicky řešitelný je, je řešitelný nějakým prakticky použitelným algoritmem? I když totiž z hlediska úvah o algoritmické řešitelnosti je zásadní rozdíl mezi výpočtem, který skončí pro provedení 100^{100} operací, a mezi nekonečným výpočtem, je v perspektivě praktické použitelnosti takový rozdíl spíše akademický.

Z úsilí podrobněji rozlišit algoritmicky řešitelné problémy na problémy více či méně prakticky zvládnutelné vyrostla tzv. teorie výpočtové složitosti. Popišme alespoň v základních rysech metodu, která se užívá při klasifikaci problémů z hlediska složitosti. Pro jednoduchost a také v souladu se zaměřením této knihy se omezíme na jediný typ problémů – rozpoznávání jazyků.

Složitost problémů se vždy testuje vzhledem k jisté předem zvolené třídě **S** výpočetních prostředků (abstraktních modelů počítačů, Turingových strojů, kombinačních sítí apod.). Pro tuto třídu **S** se dále zvolí jistá míra složitosti výpočtů **M**, která každému výpočtu c libovolného stroje $\mathscr{M} \in \mathbf{S}$ nad libovolným vstupním slovem w přiřadí jisté přirozené číslo $\mathbf{M}(\mathscr{M}, w, c)$. Nejobvyklejšími příklady takových měr je doba trvání výpočtu c stroje $\mathscr{M}$ nad w (počet vykonaných operací) nebo např. množství paměti použité při tomto výpočtu.

Pro nedeterministické stroje je pak definována *složitost přijetí slova* w *strojem* $\mathscr{M} \in \mathbf{S}$ takto:

$$\mathbf{M}_1(\mathscr{M}, w) = \min \{\mathbf{M}(\mathscr{M}, w, c);\ c \text{ je přijímající výpočet stroje } \mathscr{M} \text{ nad } w\}.$$

V případě, že $\mathscr{M}$ nepřijímá w, není $\mathbf{M}_1(\mathscr{M}, w)$ definováno. Pro deterministické $\mathscr{M}$ je prostě

$$\mathbf{M}_1(\mathscr{M}, w) = \mathbf{M}(\mathscr{M}, w, c), \text{ jestliže } \mathscr{M} \text{ přijímá } w \text{ výpočtem } c,$$

$$\mathbf{M}_1(\mathscr{M}, w) \text{ není definováno, jestliže } \mathscr{M} \text{ nepřijímá } w.$$

Jestliže f je libovolná funkce z množiny přirozených čísel do množiny přirozených čísel a $\mathscr{M}$ je stroj přijímající jazyk L, říkáme, že *stroj* $\mathscr{M}$ *přijímá jazyk* L *s* **M***-složitostí nejvýše* f, jestliže pro každé $w \in L$ je $\mathbf{M}_1(\mathscr{M}, w) \leqq f(|w|)$.

V tomto smyslu se např. říká, že určitý Turingův stroj rozpoznává symetrii slov s kvadratickou časovou složitostí, tzn. každé symetrické slovo délky n přijme během nejvýše n^2 kroků. Tímto způsobem je tedy vyjadřována složitost jednotlivých algoritmů (strojů). Zcela přirozeně lze přejít ke stanovení složitosti jazyků vzhledem ke zvolené třídě **S** a míře složitosti **M**.

Říkáme, že *jazyk* L *má* **M***-složitost nejvýše* f, jestliže existuje $\mathscr{M} \in \mathbf{S}$ rozpoznávající L s **M**-složitostí nejvýše f.

V tomto smyslu je třeba rozumět např. tvrzení, že každý kontextový jazyk je na nedeterministických Turingových strojích rozpoznatelný s lineární prostorovou složitostí. To znamená, že ke každému kontextovému jazyku L existuje nedeterministický Turingův stroj $\mathscr{M}$ a konstanta $c > 0$ tak, že každé slovo $w \in L$ je strojem $\mathscr{M}$ přijato nějakým výpočtem, který nepoužije více než $c|w|$ políček pásky.

Teorie výpočtové složitosti zaznamenala v uplynulých letech pozoruhodný rozvoj. Díky ní se podařilo najít efektivnější metody řešení mnoha důležitých problémů nebo naopak prokázat, že daný problém je natolik složitý, že jej v praxi nelze v plné

obecnosti zvládnout. Podrobnější výklad celé problematiky je možné najít např. v knize [2].

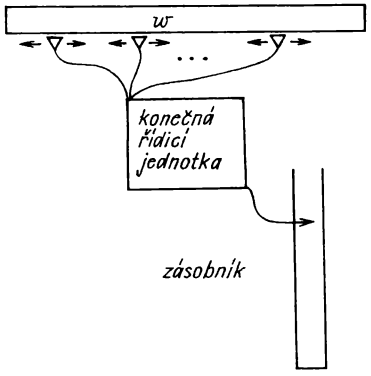
Teorii formálních jazyků nabízí studium výpočtové složitosti možnost klasifikovat jazyky z hlediska jejich složitosti (o tom je pěkná úvodní partie např. v knize [20]).

Okruh otázek probíraných v této knize vyžaduje alespoň se zmínit také o užitečné možnosti charakterizovat generativní (resp. rozpoznávací) sílu určitých typů gramatik (resp. automatů) v termínech výpočtové složitosti.

Důležitý a patrně nejstarší příklad poskytuje třída kontextových gramatik. Ze všech tříd Chomského hierarchie byla právě pro kontextové jazyky nejpozději nalezena odpovídající automatová charakterizace. Zatímco u ostatních tříd byly příslušné typy automatů (konečné automaty, zásobníkové automaty, Turingovy stroje) specifikovány pomocí typu paměti a způsobů přístupu k ní, což je postup pro teorii automatů typický, u kontextových gramatik se automatový protějšek podařilo nalézt až s použitím omezení složitosti výpočtů. Výsledkem je následující věta, jejíž důkaz lze nalézt např. v knize [20].

Věta 6.4.1. *Jazyk je kontextový, právě když jej lze rozpoznat nedeterministickým Turingovým strojem s lineárním prostorem.*

Jiným důležitým příkladem automatové charakterizace třídy jazyků určité složitosti jsou tzv. vícehlavé dvousměrné zásobníkové automaty. Každý takový automat sestává z konečné řídicí



Obr. 46

jednotky, vstupní pásky s konečným počtem čtecích hlav (nemohou přepisovat symboly), které se mohou pohybovat v obou směrech, a zásobníkové paměti (viz obr. 46).

Věta 6.4.2. *Vícehlavé dvousměrné zásobníkové automaty rozpoznávají právě jazyky, které jsou rozpoznatelné deterministickými Turingovými stroji v polynomiálním čase.*

V podrobnější formulaci: libovolný jazyk L je rozpoznatelný vícehlavým dvousměrným zásobníkovým automatem, právě když existuje deterministický Turingův stroj $\mathcal{M}$ a přirozené číslo k tak, že

1. $\mathcal{M}$ rozpoznává L,
2. pro každé $w \in L$ má výpočet $\mathcal{M}$ nad w nejvýše $|w|^k$ kroků.

Tato věta byla dokázána v článku [15].

Třída jazyků rozpoznatelných deterministickými Turingovými stroji v polynomiálním čase vyžaduje podrobnější komentář. Turingovy stroje nejsou pro popis této třídy jazyků podstatné. Při volbě libovolného jiného deterministického modelu počítače (s jednou centrální řídicí jednotkou) a ohraničení jejich výpočtů polynomiální časovou složitostí dostaneme tutéž třídu jazyků. Proto je pojem jazyka rozpoznatelného s polynomiální časovou složitostí v širokém smyslu slova invariantní vůči volbě typu stroje. To je také jeden z důvodů, proč je tento pojem často chápán jako matematický protějšek vágního pojmu jazyka rozpoznatelného pomocí prakticky realizovatelného výpočtu.

Význam věty 6.4.2 tkví v tom, že zmíněnou třídu jazyků umožňuje charakterizovat způsobem nezávislým na pojmu výpočtové složitosti.

ZÁKLADNÍ ČÁST

U všech typů automatů, s nimiž jsme měli příležitost se seznámit v průběhu předchozích kapitol, je možné pozorovat jisté společné rysy. Každý automat měl konečnou řídicí jednotku, která jednak získává informace ze vstupní pásky, jednak komunikuje s určitým typem vnější paměti. Vzájemné odlišnosti jednotlivých typů automatů jsou pak dány jednak různými typy paměti (např. zásobník u zásobníkových automatů oproti pásce u Turingových strojů), jednak způsobem přístupu ke vstupní informaci (např. jednosměrná páska u konečných automatů či zásobníkových automatů oproti dvousměrné pásce u dvousměrných konečných automatů nebo pásce čtené několika dvousměrnými hlavami u konečných mnohohlavých automatů). Tato základní strukturální podobnost ještě výrazněji vystoupí, rozhlédneme-li se po dalších druzích automatů studovaných v literatuře. Například změnou vstupní pásky na dvousměrnou vzniká ze zásobníkového automatu poměrně podrobně prozkoumaný dvousměrný zásobníkový automat (čl. [15]); rozlišením vstupní a pracovní pásky vznikají Turingovy stroje spřažené (s jednosměrnou vstupní páskou) či nespřažené (s dvousměrnou vstupní páskou). Rozšířením jejich paměti na několik pracovních pásek vznikají vícepáskové Turingovy stroje (základní údaje viz např. kniha [20]), rozšířením paměti jednopáskových Turingových strojů o zásobník vznikají tzv. automaty s pomocným zásobníkem (viz čl. [7]) atd. Další typy automatů jsou charakterizovány vnějšími paměťmi ve formě různě modifikovaných zásobníků (viz čl. [16]).

O tuto příbuznost různých druhů automatů užívaných v teorii formálních jazyků se opírají pokusy o vytvoření obecné teorie automatů (kniha [13]), zkoumání vlastností společných automatům s dvousměrnou vstupní páskou (viz práce [8]) apod.

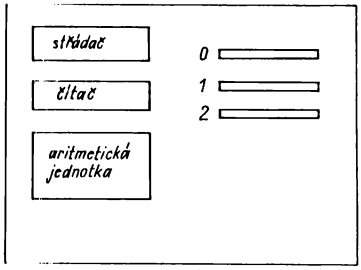
Obrátíme-li však pozornost k automatům určeným ke zpracování jiných objektů než řetězů symbolů, naskýtá se ještě mnohem pestřejší obraz. V této kapitole se soustředíme na tři takové typy automatů: stroje s přímým přístupem, stromové automaty a celulární automaty.

7.1. Stroje RASP

Stroje RASP (z anglického random access, stored program) se liší od automatů probíraných v předchozích kapitolách ve dvou rysech:

a) místo s řetězy symbolů pracují s čísly;

b) jsou výstižnějším obrazem současných počítačů.



Obr. 47

Každý stroj RASP se skládá (viz obr. 47) z aritmetické jednotky, střádače, čítače a neomezeného množství registrů ohodnocených (adresovaných) nezápornými celými čísly. Střádač, čítač i každý z registrů mohou obsahovat libovolné celé nezáporné číslo. Obsah střádače, čítače i jednotlivých registrů mění aritmetická jednotka tak, že provádí program tvořený instrukcemi ze souboru instrukcí uvedených v tab. 16. Provedením instrukce LOAD *i* se obsah *i*-tého registru okopíruje do střádače (obsah *i*-tého registru zůstane nezměněn), provedením instrukce LOAD = *i* se číslo *i* přenese do střádače. Instrukcí STORE *i* se obsah střádače (aniž se změní) okopíruje do registru *i*. Instrukce ADD *i* (resp. SUB *i*, MULT *i*, DIV *i*) přičte k obsahu střádače obsah registru *i* (resp. odečte, vynásobí, celočíselně vydělí). Obsah registru *i* se opět nemění. Instrukce ADD = *i* (resp. SUB = *i*,

MULT = i, DIV = i) přičte k obsahu střádače číslo i (resp. odečte, vynásobí, celočíselně vydělí). Instrukce HALT ukončuje výpočet.

Tak např. jednoduchý program složený z instrukcí

(*)
```
LOAD 0,
ADD = 1,
MULT 1,
STORE 0,
HALT
```

vynásobí obsah registru 0 zvětšený o jedničku obsahem registru 1 a výsledek uloží do registru 0.

Program pro RASP i data pro výpočet jsou na začátku výpočtu uloženy v registrech. Každá instrukce je uložena ve dvou po sobě jdoucích registrech. V prvním z nich je uložen kód instrukce (viz tab. 16), ve druhém hodnota operandu.

Symbolický zápis instrukce	Kód instrukce
LOAD i	1
LOAD = i	2
STORE i	3
ADD i	4
ADD = i	5
SUB i	6
SUB = i	7
MULT i	8
MULT = i	9
DIV i	10
DIV = i	11
JUMP i	12
JGTZ i	13
JZERO i	14
HALT	15

Tab. 16

Například program (*) spolu se vstupními daty může být uložen tak, jak je znázorněno v tab. 17:

Číslo registru	Obsah
0	n
1	m
2	1
3	0
4	5
5	1
6	8
7	1
8	3
9	0
10	15
11	0

Tab. 17

V prvních dvou registrech jsou uložena vstupní data (čísla m, n), v registrech 2, 3 je zakódována instrukce LOAD 0, v registrech 4, 5 instrukce ADD = 1 atd.

Adresa prováděné instrukce je uložena v čítači. Po provedení libovolné z instrukcí s kódy 1 až 11 se obsah čítače zvětší o dvě, při zastavení výpočtu instrukcí HALT se obsah čítače nemění. To znamená, jestliže je počáteční obsah registrů jako v tab. 17 a počáteční hodnota čítače je nastavena na 2, provede se celý program (*) a na jeho konci bude obsah registru 0 roven $(n + 1)m$, obsah ostatních registrů bude nezměněn a v čítači bude adresa 10.

Pravidelné zvětšování obsahu čítače lze přerušit skokovými instrukcemi (kód 12 až 14). Provedení instrukce JUMP i spočívá v nastavení čítače na hodnotu i, při provedení JGTZ i (jump if greater then zero) se čítač nastaví na i v případě, že obsah střádače je větší než 0, v opačném případě se obsah čítače zvětší o 2. Instrukce JZERO i (jump if zero) nastaví čítač na i v případě, že obsah střádače je roven 0, v opačném případě se obsah čítače zvětší o 2.

Jestliže dojde k situaci, že čítač obsahuje adresu registru, v němž

je uloženo číslo, které nelze interpretovat jako kód instrukce (tzn různé od 1 až 15), další krok není definován.

Příklad 7.1.1. (program pro výpočet faktoriálu). Jestliže obsah registrů stroje RASP je jako v tab. 18, n je kladné celé číslo a po-

Adresa	Obsah registru	
0	n	
1	1	LOAD 0
2	0	
3	14	JZERO 21
4	21	
5	7	SUB = 1
6	1	
7	14	JZERO 21
8	21	
9	3	STORE 23
10	23	
11	1	LOAD 0
12	0	
13	8	MULT 23
14	23	
15	3	STORE 0
16	0	
17	1	LOAD 23
18	23	
19	12	JUMP 5
20	5	
21	15	HALT
22	0	
23	0	
24	0	
25	0	
⋮	⋮	

Tab. 18

počáteční hodnotu čítače je 1, potom proběhne výpočet, po jehož ukončení bude v registru 0 číslo $n!$.

Do registru 23 jsou v průběhu výpočtu postupně ukládána čísla n, $n-1$, $n-2$, $n-3$, ..., kterými je postupně násoben obsah registru 0. Čtenář, který není obeznámen s programováním počítače ve strojovém kódu, může s užitkem na papíře simulovat průběh výpočtu programu z tab. 18 pro nějaké konkrétní číslo n.

Příklad naznačuje, jakým způsobem je možné stroje RASP používat k výpočtu funkcí (i více proměnných). Stačí specifikovat uložení programu, registry obsahující vstupní data, registry obsahující výsledek a počáteční hodnotu čítače.

	Adresa	Obsah registru	
LOAD = 1	0	n	
STORE 2	1	m	
STORE 5	2	1	LOAD 0
STORE 7	3	0	
LOAD = 0	4	5	ADD = 1
STORE 3	5	1	
STORE 9	6	8	MULT 1
STORE 11	7	1	
LOAD = 5	8	3	STORE 0
STORE 4	9	0	
LOAD = 8	10	15	HALT
STORE 6	11	0	
LOAD = 3			- - - - - -
STORE 8	12	3	STORE 3
LOAD = 15	13	3	
STORE 10	14	3	STORE 9
JUMP 2	15	9	
	⋮	⋮	
a)		b)	

Tab. 19

Skutečnost, že data nejsou ve strojích RASP oddělena od programu, umožňuje např. měnit program v průběhu výpočtu. Uvedeme jednoduchý příklad.

Příklad 7.1.2. Nechť v paměti stroje RASP je počínaje druhým registrem uložen program z tab. 19a) a v prvních dvou registrech libovolná čísla m, n. Nechť výpočet začne s čítačem nastaveným na 2.

Na první pohled se zdá, že tento program ani nemůže úspěšně skončit, protože neobsahuje instrukci HALT. Při bližším zkoumání se však ukáže, že jde o korektní program, který se zastaví a v okamžiku ukončení má v registru 0 číslo $(n + 1)m$. Provedením programu až k instrukci STORE 10 se totiž můžeme přesvědčit, že pro provedení této instrukce má počáteční úsek paměti obsah jako je znázorněno v tab. 19b). Pro provedení následující instrukce JUMP 2 je čítač nastaven na 2. Segment 2 až 11 ale obsahuje program pro výpočet $(n + 1)m$ (srov. tab. 17), který končí instrukcí HALT.

Jedna z nejdůležitějších možností, které nabízí taková sebemodifikace programů, je tzv. nepřímá adresace. Při ní je možné nejen dosáhnout registru daného adresou i, ale i registru, jehož adresa je dána číslem obsaženým v registru i. Označme obsah registru i symbolem $c(i)$. Potom např. úsek programu

adresa	obsah	
k	1	LOAD i
$k+1$	i	
$k+2$	3	STORE $k+5$
$k+3$	$k+5$	
$k+4$	1	LOAD 0
$k+5$	0	

realizuje přenesení obsahu registru $c(i)$ do střádače. Po provedení instrukcí LOAD i a STORE $k + 5$ je totiž v registrech $k + 4$,

$k + 5$ kód instrukce LOAD $c(i)$. Podobně lze realizovat STORE $c(i)$, ADD $c(i)$, JUMP $c(i)$ atd.

Právě možnost nepřímého adresování odlišuje podstatně možnosti přístupu k paměti u strojů RASP, např. od Turingových strojů. Zatímco u Turingových strojů lze jedním krokem přejít pouze na sousední políčko, umožňuje RASP v registru rychle vytvořit pomocí aritmetických operací i adresu velmi vzdáleného registru a ten pak dosáhnout nepřímou adresací.

Tím zajímavější je zjištění, že stroje RASP jsou v jistém smyslu ekvivalentní Turingovým strojům. O tom pojednává čl. 7.2.

7.2. Turingova teze

V článku [30], ve kterém navrhl formalizaci pojmu algoritmu pomocí abstraktních strojů, vyjádřil A. M. Turing také názor, že jím navržená formalizace skutečně dobře vystihuje intuitivní pojem algoritmu. Jeho tezi bychom mohli vyjádřit těmito slovy:

Každý Turingův stroj reprezentuje nějaký algoritmus a obráceně, každý algoritmicky řešitelný problém lze řešit vhodným Turingovým strojem.

Uvedená teze není matematickou větou, protože hovoří o algoritmech v nikoli přesně matematicky definovaném smyslu. Proto ji také nelze dokazovat, ale pouze ověřovat její věrohodnost. Argumenty, které svědčí v její prospěch, jsou natolik přesvědčivé, že je bez výhrady přijímána.

Jeden typ argumentů se zabývá tím, že Turingovy stroje nevybočují z třídy postupů, obecně uznávaných za algoritmické, a naopak, že ke každému dosud navrženému postupu souhlasně považovanému za algoritmický je možné ověřit existenci ekvivalentního Turingova stroje.

Snad ještě přesvědčivějším typem argumentu je skutečnost, že všechny matematické formalizace algoritmu se ukázaly ekvivalentní Turingovým strojům, jakkoli odlišné bylo jejich východisko. Příkladem, se kterým jsme se již setkali v této knize (6.1.10), je ekvivalence gramatik typu 0 s Turingovými stroji.

V této souvislosti se vynoří přirozená otázka. Turingovy stroje používáme jako prostředky k rozpoznávání jazyků a je možné

jimi reprezentovat transformace řetězů, tzn. jistá zobrazení z množiny slov do množiny slov. Pojem algoritmu jsme ovšem zvyklí spojovat s rozmanitějšími objekty – čísly, maticemi, grafy apod. Jak je možné přirozeným způsobem definovat a prokázat ekvivalenci obecných algoritmů nad takovými objekty a Turingových strojů?

Odpověď je obsažena v možnosti kódovat takové objekty pomocí řetězů symbolů. Místo se skutečnými objekty pracuje pak Turingův stroj s jejich kódy. Jelikož také příslušné objekty naopak poskytují možnost kódovat řetězy symbolů, lze pomocí algoritmů nad takovými objekty simulovat činnost Turingových strojů.

V tomto smyslu lze např. dokázat ekvivalenci Turingových strojů se stroji RASP. Ukážeme aspoň v hrubých rysech, jak by se jedna část takového důkazu vedla.

Nejdříve je třeba zvolit kódování čísel pomocí řetězů v nějaké pevně dané abecedě Σ a obráceně. Z mnoha možností, které se nabízejí, zvolíme následující kódování K řetězů pomocí čísel.

Nechť Σ je n-prvková abeceda. Přiřaďme jednotlivým symbolům vzájemně jednoznačně čísla $1, \ldots, n$, tzn.

$$K(a) \in \{1, \ldots, n\}$$

pro každé $a \in \Sigma$ a $K(a) \neq K(a')$ pro $a \neq a'$.

Ostatní slova kódujme takto:

$$K(e) = 0,$$

$$K(a_k \ldots a_1) = \sum_{j=1}^{k} K(a_j)\, n^j$$

pro libovolná $a_1, \ldots, a_k \in \Sigma$, $k \geqq 1$.

Jedná se o tzv. *n-adický zápis čísel.*

Příklad 7.2.1. Nechť abeceda $\Sigma = \{a, b\}$. Položme $K(a) = 1$, $K(b) = 2$. Potom slova z $\{a, b\}^*$ jsou kódována takto:

$$K(e) = 0,$$
$$K(a) = 1,$$
$$K(b) = 2,$$

$$K(aa) = 3,$$
$$K(ab) = 4,$$
$$K(ba) = 5,$$
$$K(bb) = 6,$$
$$K(aaa) = 7,$$
$$\vdots$$

To znamená ztotožníme-li a s 1, b s 2, dostáváme dyadický zápis čísel:

1 ... 1,
2 ... 2,
3 ... 11,
4 ... 12,
5 ... 21,
6 ... 22,
7 ... 111 atd.

Tímto kódováním se množina Σ^* vzájemně jednoznačně zobrazuje na množinu nezáporných celých čísel.

Nyní už je možné pohlížet na stroje RASP jako na prostředky pro práci s řetězy. V souhlase s Turingovou tezí lze dokázat, že stroje RASP jsou schopny rozpoznávat přesně tutéž třídu jazyků (resp. realizovat tytéž funkce) jako Turingovy stroje.

Pro zajímavost probereme jednu z mnoha možností, jak mohou stroje RASP simulovat činnost Turingových strojů.

Turingovskou pásku obsahující řetěz uv, s hlavou na prvním symbolu slova v, lze reprezentovat dvěma registry, z nichž jeden obsahuje číslo $K(u)$ a druhý číslo $K(v^R)$. Při simulování pohybu hlavy o políčko doprava se umaže poslední symbol řetězu v^R a připíše se na konec u. Podobně při simulaci pohybu hlavy doleva. Takovou operaci lze úspěšně provést, jakmile je možné realizovat tyto základní úkony:

a) testování, jaký je poslední symbol řetězu zakódovaného jako číslo;

b) přechod od čísla kódujícího jistý řetěz k číslu kódujícímu řetěz vzniklý vymazáním posledního symbolu původního řetězu

c) přechod od čísla kódujícího řetěz k číslu kódujícímu řetěz vzniklý z původního řetězu připsáním daného symbolu zprava

Všechny tři úkony je možné na strojích RASP provést vhodnými aritmetickými operacemi.

Příklad 7.2.2. V případě kódování dvojprvkové abecedy z př. 7.2.1 se zmíněné tři úkony provedou takto:

a) Testování posledního písmene v řetězu kódovaném číslem $k > 0$: jestliže $k = 2 \times (k \div 2)$ (kde $\div$ označuje celočíselné dělení), potom poslední symbol řetězu je b, v opačném případě a.

b) Vymazání posledního písmene se realizuje jako přechod od čísla k k zaokrouhlené hodnotě čísla $(2k - 1)/4$.

c) Připsání a zprava se realizuje jako přechod od k k $2k + 1$, připsání b jako přechod od k k $2k + 2$.

Z toho už je vidět, jak by se dokázalo tvrzení, že každý Turingův stroj lze simulovat vhodně naprogramovaným strojem RASP, který nahrazuje turingovskou pásku dvěma registry.

Důkaz opačného tvrzení, totiž že každý stroj RASP lze simulovat vhodným Turingovým strojem, je už o něco náročnější.

S důkazy obou tvrzení je možné se podrobně obeznámit např. v knize [2].

ROZŠIŘUJÍCÍ ČÁST

7.3. Stromové automaty

Dalším typem automatů pracujících nad jinými objekty než řetězy jsou tzv. stromové automaty. Nejčastěji uvažovaným typem stromových automatů jsou automaty zpracovávající stromy shora dolů (tj. od kořene k listům). Přitom se zpravidla jedná o tzv. ohodnocené uspořádané stromy.

Definice 7.3.1. *Ohodnocený uspořádaný strom* se zadává jako pětice

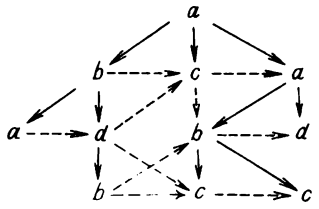
$$\mathscr{T} = (V, H, <, \Sigma, f),$$

kde V je konečná neprázdná množina (vrcholů), $H \subseteq V \times V$ je

množina hran, $< \subseteq V \times V$ je částečné uspořádání, Σ je konečná abeceda, $f: V \to \Sigma$ je ohodnocení vrcholů a jsou splněny tyto podmínky:

1. Dvojice (V, H) je strom, tzn. existuje vrchol $v_0 \in V$ (kořen stromu) takový, že pro každý vrchol v různý od v_0 existuje právě jedna cesta z v_0 do v, tzn. právě jedna posloupnost vrcholů $v_1, \ldots, v_n$ $(n \geq 1)$ taková, že $(v_i, v_{i+1}) \in H$ pro všechna i, $0 \leq i < n$ a $v_n = v$.

2. Libovolné dva vrcholy v, v' jsou srovnatelné v uspořádání $<$, právě když ve stromu (V, H) nevede cesta od v k v' ani od v' k v.



Obr. 48

Příkladem ohodnocených uspořádaných stromů jsou např. stromy popisující strukturu řetězu podle gramatiky, derivační stromy (viz 4.2.4 a 4.2.5) nebo strom z obr. 48, u něhož

$$\Sigma = \{a, b, c, d\},$$

hrany z H jsou vyznačeny plně a uspořádání $<$ odpovídající uspořádání zleva doprava je dáno jako reflexivní a tranzitivní uzávěr relace vyznačené čárkovanými šipkami.

V dalším výkladu budeme uspořádání $<$ reprezentovat v obrázcích ohodnocených uspořádaných stromů uspořádáním zleva doprava podobně jako na obr. 48.

V tomto článku budeme dále místo ohodnocený uspořádaný strom říkat pouze strom.

K popisu činnosti stromových automatů se s výhodou využívá lineární reprezentace stromů, kterou zavedeme induktivně na základě jednoho pomocného pojmu.

Definice 7.3.2. Nechť $\mathscr{T} = (V, H, <, \Sigma, f)$ je strom a $v \in V$. *Podstromem stromu* $\mathscr{T}$ určeným vrcholem v nazýváme strom

$\mathscr{T}_v = (V', H', <', \Sigma, f')$, kde $V' = \{v\} \cup \{\bar{v};$ v $\mathscr{T}$ vede cesta z v do $\bar{v}\}$ a H', $<'$, f' vzniknou po řadě z H, $<$ a f parcializací na množinu V'.

Definice 7.3.3. Nechť $\mathscr{T} = (V, H, <, \Sigma, f)$ je strom a], [symboly nepatřící do abecedy Σ. Potom pojem kódu $K(\mathscr{T})$ stromu $\mathscr{T}$ definujeme induktivně takto:

1. Jestliže $|V| = 1$, tzn. $V = \{v_0\}$, potom $K(\mathscr{T}) = f(v_0)$.
2. Jestliže $v_1 < v_2 < \ldots < v_k$ $(k \geqq 1)$ jsou všechny vrcholy, do nichž vchází nějaká hrana vycházející z kořene v_0 stromu $\mathscr{T}$, potom

$$K(\mathscr{T}) = f(v_0)\,[K(\mathscr{T}_{v_1})\,K(\mathscr{T}_{v_2}) \ldots K(\mathscr{T}_{v_k})]\,.$$

Příklad 7.3.4. Kódem stromu z obr. 48 je řetěz

$$a[b[ad[b]]\,ca[b[cc]\,d]]\,.$$

Definice 7.3.5. *Stromový automat* je definován jako šestice

$$\mathscr{A} = (Q, \Sigma, \Delta, \Pi, q_0, P)\,,$$

kde Q je konečná množina stavů, Σ je konečná vstupní abeceda, Δ je konečná výstupní abeceda, Π je konečná abeceda proměnných, $q_0 \in Q$ je počáteční stav a P je konečná množina pravidel tvaru

$$q(a[x_1 \ldots x_n]) \to w_1 q_1(x_{i_1})\, w_2 q_2(x_{i_2}) \ldots q_r(x_{i_r})\, w_{r+1}\,,$$

kde

$$n \geqq 0\,,\ r \geqq 0\,,\ q, q_1, \ldots, q_r \in Q\,,\ a \in \Sigma\,,\ w_1, \ldots, w_{r+1} \in \Delta^*\,,$$

$$1 \leqq i_j \leqq n \text{ pro všechna } j \leqq r\,,\ x_1, \ldots, x_n \in \Pi\,.$$

Automat $\mathscr{A}$ se nazývá *deterministický*, jestliže různá pravidla z P mají různé levé strany.

Automat zpracovává kód libovolného stromu $\mathscr{T}$ takto:

1. zahájí činnost s řetězem $q_0(K(\mathscr{T}))$;
2. v každém kroku nahradí v přepisovaném slově všechny pod-

řetězy tvaru $q(K(\mathscr{T}_v))$, kde $q \in Q$, podle některého pravidla P takto: jestliže

$$K(\mathscr{T}_v) = a[K(\mathscr{T}_1) \dots K(\mathscr{T}_n)]$$

pro nějaké $a \in \Sigma$ a $n \geqq 0$ a jestliže P obsahuje pravidlo

$$q(a[x_1 \dots x_n]) \to w_1 q_1(x_{i_1}) \dots q_r(x_{i_r}) w_{r+1},$$

potom $q(K(\mathscr{T}_v))$ je možné nahradit řetězem

$$w_1 q_1 K(\mathscr{T}_{i_1}) \dots q_r K(\mathscr{T}_{i_r}) w_{r+1}.$$

Příklad 7.3.6. Deterministický automat s množinou stavů

$$Q = \{q_0, q_1\}$$

(q_0 je počáteční stav), vstupní abecedou

$$\Sigma = \{a, b, c, d\},$$

výstupní abecedou

$$\Delta = \{0, 1, 2,], [\},$$

množinou proměnných

$$\Pi = \{x_1, x_2, x_3\}$$

a množinou pravidel

(1) $q_0(a[x_1 x_2 x_3]) \to 0[q_0(x_1)\, q_0(x_3)\, q_1(x_3)]$,

(2) $q_0(a[x_1 x_2]) \to 1[2q_1(x_2)]$,

(3) $q_1(a[x_1 x_2]) \to 2[q_0(x_1)\, q_1(x_2)]$,

(4) $q_0(b[x_1 x_2]) \to 0[q_1(x_1)\, q_1(x_1)]$,

(5) $q_1(a) \to 2$,

(6) $q_1(b) \to 1$,

(7) $q_1(c) \to 1$,

(8) $q_1(d) \to 0$

zpracovává strom z obr. 48 takto (viz též obr. 49):

$$q_0(\underbrace{a[b[ad[b]]}_{K(\mathscr{T}_1)}\ \underbrace{c}_{K(\mathscr{T}_2)}\ \underbrace{a[b[cc]\,d]]}_{K(\mathscr{T}_3)}) \vdash$$

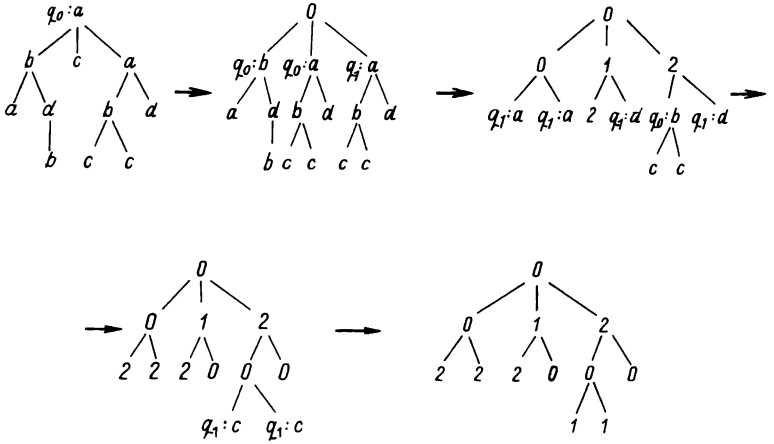
$$\vdash 0[q_0(b[ad[b]])\,q_0(a[b[cc]\,d])\,q_1(a[b[cc]\,d])] \vdash$$

$$\vdash 0[0[q_1(a)\,q_1(a)]\,1[2q_1(d)]\,2[q_0(b[cc])\,q_1(d)]] \vdash$$

$$\vdash 0[0[22]\,1[20]\,2[0[q_1(c)\,q_1(c)]\,0]] \vdash$$

$$\vdash 0[0[22]\,1[20]\,2[0[11]\,0]]\,.$$

Každý strom určuje jisté slovo, které je tvořeno ohodnocením jeho listů.



Obr. 49

Definice 7.3.7. Nechť $\mathscr{T} = (V, H, <, \Sigma, f)$ je libovolný strom a $v_1 < v_2 < \ldots < v_n$ všechny jeho listy (tj. vrcholy, z nichž nevychází žádná hrana). *Hraniční slovo* $H(\mathscr{T})$ stromu $\mathscr{T}$ definujeme jako $H(\mathscr{T}) = f(v_1)\,f(v_2)\ldots f(v_n)$.

Příklad 7.3.8. Hraničním slovem stromu z obr. 48 je řetěz *abcccd*.

Zajímavá souvislost existuje mezi třídou bezkontextových jazyků a třídou definičních oborů stromových automatů.

Definice 7.3.9. Pro každý stromový automat

$$\mathscr{A} = (Q, \Sigma, \Delta, \Pi, q_0, P)$$

definujeme jeho definiční obor $D(\mathscr{A})$ jako množinu těch stromů $\mathscr{T}$, u kterých $\mathscr{A}$ může převést slovo $q_0(K(\mathscr{T}))$ na nějaký řetěz neobsahující žádný symbol z Q.

Příklad 7.3.10. Strom z obr. 48 patří do definičního oboru automatu $\mathscr{A}$ z př. 7.3.6, zatímco strom s kódem $a[b[baa]\,ca]$ do jeho definičního oboru nepatří, neboť

$$q_0(a[b[baa]\,ca]) \vdash_{\mathscr{A}} 0[q_0(b[baa])\,q_0(a)\,q_1(a)]$$

a toto slovo nelze dále přepsat, protože pravidla automatu $\mathscr{A}$ neumožňují přepsat $q_0(b[baa])$.

Věta 7.3.11. *Každý bezkontextový jazyk neobsahující prázdné slovo je množinou hraničních slov definičního oboru nějakého stromového automatu.*

Důkaz. Každý bezkontextový jazyk L neobsahující prázdné slovo lze podle 3.2.8 generovat nějakou nevypouštějící gramatikou $\mathscr{G} = (\Pi, \Sigma, S, P)$. Sestrojme stromový automat $\mathscr{A}$, jehož definiční obor je tvořen právě derivačními stromy podle gramatiky $\mathscr{G}$, a tím bude věta dokázána. Vezměme z P pravidlo s nejdelší pravou stranou a označme její délku k.

Definujme

$$\mathscr{A} = (\Pi \cup \Sigma, \Pi \cup \Sigma, \{a\}, \{x_1, \ldots, x_k\}, S, R),$$

kde a je libovolný nově přidaný symbol a množina pravidel R je definována takto: pro libovolné

$$X \in \Pi,\ r \leqq k,\ Y_1, \ldots, Y_r \in \Pi \cup \Sigma$$

je

$$X(X[x_1 \ldots x_r]) \to a[Y_1(x_1) \ldots Y_r(x_r)] \in R,$$

právě když $X \to Y_1 \ldots Y_r \in P$.

Pro každé $c \in \Sigma$ je v R pravidlo $c(c) \to a$.

Jiná pravidla R neobsahuje. Čtenář sám snadno ověří indukcí podle počtu vnitřních vrcholů stromu (tj. vrcholů, z nichž vy-

chází aspoň jedna hrana), že definiční obor $\mathscr{A}$ obsahuje právě derivační stromy podle $\mathscr{G}$.

Příklad 7.3.12. Derivační stromy podle gramatiky

$$S \to 0S1A \mid 0,$$
$$A \to SA \mid 1$$

tvoří definiční obor stromového automatu

$$(\{S, A, 0, 1\}, \{S, A, 0, 1\}, \{a\}, \{x_1, x_2, x_3, x_4\}, S, R),$$

kde R zahrnuje právě pravidla

$$\begin{aligned}
&S(S[x_1x_2x_3x_4]) \to a[0(x_1)\, S(x_2)\, 1(x_3)\, A(x_4)],\\
&S(S[x_1]) \to a[0(x_1)],\\
&A(A[x_1x_2]) \to a[S(x_1)\, A(x_2)],\\
&A(A[x_1]) \to a[1(x_1)],\\
&0(0) \to a,\\
&1(1) \to a.
\end{aligned}$$

Věta 7.3.13. *Množina hraničních slov definičního oboru libovolného stromového automatu tvoří bezkontextový jazyk neobsahující prázdné slovo.*

Důkaz. Nechť $\mathscr{A} = (Q, \Sigma, \Delta, \Pi, q_0, P)$ je libovolný stromový automat. Sestrojíme nevypouštějící bezkontextovou gramatiku $\mathscr{G}$ generující množinu hraničních slov definičního oboru automatu $\mathscr{A}$. Množinou terminálů gramatiky $\mathscr{G}$ je abeceda Σ. Množina neterminálů je rovna $\mathscr{P}(Q)$, počátečním neterminálem je množina $\{q_0\}$. Zbývá popsat množinu R přepisovacích pravidel gramatiky $\mathscr{G}$. Tato množina se bude skládat z těchto dvou typů pravidel:

1. Pro libovolné $N \subseteq Q$ a $a \in \Sigma$ je

$$N \to a \in R,$$

právě když ke každému $q \in N$ existuje $u \in \Delta^*$ tak, že

$$q(a) \to u \in P.$$

(Speciálně pro $\emptyset$ obsahuje R pravidla $\emptyset \to a$ pro všechna $a \in \Sigma$.)

2. Nechť $N \subseteq Q$, $a \in \Sigma$ a přirozené číslo $n \leqq |\Pi|$ jsou zvoleny tak, že pro každé $q \in N$ existuje v P pravidlo s levou stranou $q(a[x_1 \dots x_n])$. Budiž nyní $P_1 \subseteq P$ libovolná množina taková, že ke každému $q \in N$ existuje právě jedno pravidlo z P_1 s levou stranou $q(a[x_1 \dots x_n])$. Pro každou takovou množinu P_1 bude množina R obsahovat pravidlo $N \to N_1 \dots N_n$ sestrojené následujícím způsobem.

Pro všechna i, $1 \leqq i \leqq n$, definujme

$$N_i = \{q;\ (\exists \hat{q} \in N)\, \hat{q}(a[x_1 \dots x_n]) \to uq(x_i)\, v \in P_1 \text{ pro nějaká slova } u, v]\}\,.$$

(Speciálně pro $N = \emptyset$ obsahuje R všechna pravidla typu $\emptyset \to \emptyset^n$, kde $n \leqq |\Pi|$. Tato pravidla lze ovšem nahradit jediným pravidlem $\emptyset \to \emptyset\emptyset$. V každém případě neterminál $\emptyset$ generuje celé Σ^+.)

Jiná pravidla, než utvořená podle 1 a 2, množina R neobsahuje.

K tomu, abychom dokázali tvrzení věty, stačí ověřit, že pro každé $N \subseteq Q$ a $w \in \Sigma^+$ platí tato ekvivalence:

(*) $N \Rightarrow_{\mathscr{G}}^* w \Leftrightarrow$ existuje strom $\mathscr{T}$ s hraničním slovem w takový, že pro každé $q \in N$ může automat $\mathscr{A}$ přepsat $q(K(\mathscr{T}))$ na řetěz neobsahující symboly z Q.

Zvolíme-li v (*) $N = \{q_0\}$, dostáváme tvrzení věty.

I. Ověřme implikaci $\Rightarrow$ ve vztahu (*). Postupujme indukcí podle délky derivace.

a) Nechť $N \Rightarrow w$ derivací délky 1. V tom případě se uplatní pravidlo typu 1 a tvrzení plyne přímo z definice těchto pravidel.

b) Nechť implikace $\Rightarrow$ ve vztahu (*) platí, jakmile odvození $N \Rightarrow^* w$ je délky nejvýše k. Dokážeme, že platí i pro $k + 1$.

Budiž tedy $N \Rightarrow^* w$ odvozením délky $k + 1$. Potom existují $N_1, \dots, N_n$ a $u_1, \dots, u_n$ tak, že

$$N \Rightarrow N_1 \dots N_n \Rightarrow^* w\,,\quad w = u_1 \dots u_n$$

a pro všechna $i \leqq n$ je $N_i \Rightarrow^* u_i$ odvozením délky nejvýše k. Pro všechna N_i, u_i tedy podle indukčního předpokladu existuje strom $\mathscr{T}_i$ s hraničním slovem u_i tak, že platí implikace $\Rightarrow$ v (*).

K pravidlu $N \to N_1 \dots N_n$ existuje $a \in \Sigma$, $n \leqq |\Pi|$ a $P_1 \subseteq P$ tak, že pro každé $q \in N$ je $q(a[x_1 \dots x_n])$ levou stranou právě jednoho pravidla z P_1.

Zvolme strom $\mathscr{T}$ daný kódem $K(\mathscr{T}) = a[K(\mathscr{T}_1) \dots K(\mathscr{T}_n)]$. Ukážeme, že $\mathscr{T}$ spolu s N a w splňuje $(*)$.

Vyberme libovolné $\hat{q} \in N$ a nechť $\hat{q}(K(\mathscr{T})) \Rightarrow \alpha$. Potom pro každé podslovo slova α tvaru $q(K(\mathscr{T}_i))$ je $q \in N_i$, podle definice N_i. Proto lze α přepsat na slovo neobsahující symboly z Q (podle indukčního předpokladu o N_i a $\mathscr{T}_i$).

II. Obrácená implikace $\Leftarrow$ ve vztahu $(*)$ se dokáže podobně, indukcí podle výšky stromu $\mathscr{T}$.

a) Pro stromy s jediným vrcholem plyne tvrzení okamžitě z definice pravidel typu 1.

b) Nechť implikace platí, jakmile je strom $\mathscr{T}$ výšky maximálně k. Zvolíme-li nyní strom $\mathscr{T}$ výšky $k+1$ s hraničním slovem w a takový, že pro každé $q \in N$ může $\mathscr{A}$ přepsat $q(K(\mathscr{T}))$ na řetěz z Δ^*, lze kód $\mathscr{T}$ psát ve tvaru

$$K(\mathscr{T}) = a[K(\mathscr{T}_1) \dots K(\mathscr{T}_n)]$$

pro nějaké $a \in \Sigma$ a stromy $\mathscr{T}_1, \dots, \mathscr{T}_n$, které mají po řadě hraniční slova $u_1, \dots, u_n$, tzn. $w = u_1 \dots u_n$. Pro každé $q \in N$ zvolme jedno pravidlo, které zahajuje některé úspěšné přepisování $q(K(\mathscr{T}))$. Tak dostaneme jistou množinu pravidel P_1 splňující podmínku z definice pravidel typu 2, a tedy i jisté pravidlo $N \to N_1 \dots N_n$ takové, že pro každé $q \in N_i$ je možné $q(K(\mathscr{T}_i))$ úspěšně přepsat (pro libovolné $i \leqq n$). Proto podle indukčního předpokladu $N_i \Rightarrow^* u_i$. Dostáváme tak $N \Rightarrow N_1 \dots N_n \Rightarrow^* w$.

Příklad 7.3.14. Ke stromovému automatu

$$\mathscr{A} = (\{q_0, q_1\}, \{a, b\}, \{0, 1\}, \{x_1, x_2, x_3\}, q_0, P)$$

s množinou pravidel

(1) $q_0(a[x_1 x_2 x_3]) \to 0[q_0(x_1)\, q_0(x_2)\, q_1(x_1)]$,

(2) $q_0(a[x_1 x_2 x_3]) \to 1[q_1(x_1)\, q_0(x_2)\, q_0(x_3)]$,

(3) $q_1(a[x_1 x_2 x_3]) \to 0[q_0(x_1)\, q_1(x_2)\, q_1(x_3)\, q_1(x_1)]$,

(4) $q_0(a) \to 0$,

(5) $q_0(b) \to 1$,

(6) $q_1(a) \to 0$

sestrojíme bezkontextovou gramatiku generující množinu hraničních slov definičního oboru automatu $\mathscr{A}$:

$$\{q_0\} \to \{q_0, q_1\}\{q_0\}\emptyset \mid \{q_1\}\{q_0\}\{q_0\} \mid a \mid b,$$

$$\{q_1\} \to \{q_0, q_1\}\{q_1\}\{q_1\} \mid a,$$

$$\{q_0, q_1\} \to \{q_0, q_1\}\{q_0, q_1\}\{q_1\}$$

[sestrojeno na základě pravidel (1), (3)] |

$$\{q_0, q_1\}\{q_0, q_1\}\{q_0, q_1\}$$

[sestrojeno na základě pravidel (2), (3)]

a

$$\emptyset \to \emptyset\emptyset \mid a \mid b.$$

Ve všech předchozích příkladech byl výsledkem činnosti automatu nad stromem (přesněji – kódem stromu) opět strom (kód stromu). Za to vděčíme jen speciálnímu tvaru pravidel zvolených automatů. V obecném případě tomu tak nemusí být. Tehdy pohlížíme na stromový automat jako na prostředek k transformaci stromů na lineární řetězy. Tak např. automat

$$\mathscr{A} = (\{q_0\}, \{a, b, c, d\}, \{a, b, c, d\}, \{x_1, x_2, x_3\}, q_0, P)$$

s množinou pravidel

$$q_0(a[x_1x_2x_3]) \to aq_0(x_1)\,aq_0(x_2)\,aq_0(x_3)\,a,$$

$$q_0(a[x_1x_2]) \to aq_0(x_1)\,aq_0(x_2)\,a,$$

$$q_0(b[x_1x_2]) \to bq_0(x_1)\,bq_0(x_2)\,b,$$

$$q_0(d[x_1]) \to dq_0(x_1)\,d,$$

$$q_0(a) \to a,$$

$$q_0(b) \to b,$$

$$q_0(c) \to c,$$

$$q_0(d) \to d$$

provede nad stromem z obr. 48 výpočet

$$q_0(a[b[ad[b]]\, ca[b[cc]\, d]]) \vdash$$
$$\vdash aq_0(a[b[ad[b]]) aq_0(c) aq_0(a[b[cc]\, d]]) a \vdash$$
$$\vdash \ldots \vdash ababdbdbacaabcbcbadaa\,,$$

jehož výsledkem je slovo, které získáme procházením stromu od kořene směrem zleva doprava, při němž se každá návštěva vrcholu zaznamenává.

Stromové automaty mají své pevné místo v teorii syntaxí řízeného překladu. Překlad z jednoho jazyka do druhého lze často zadat jako transformaci derivačních stromů zdrojového jazyka na derivační stromy cílového jazyka. Derivační strom zdrojového textu je možné získat metodami, z nichž některé jsme probrali v kap. 5. Příslušnou transformaci stromů se pak nezřídka podaří s výhodou popsat pomocí stromového automatu. Uvedeme jednoduchý příklad.

Příklad 7.3.15. Mějme dvě gramatiky

$$\begin{aligned}
\mathscr{G}_1\colon\ & E \to E + T,\\
& E \to T,\\
& T \to T * F,\\
& T \to F,\\
& F \to (E),\\
& F \to a\,;\\
\mathscr{G}_2\colon\ & E \to ET +\,,\\
& E \to T,\\
& T \to TF *,\\
& T \to F,\\
& F \to E,\\
& F \to a\,,
\end{aligned}$$

kde E, T, F jsou neterminály, E je počáteční symbol.

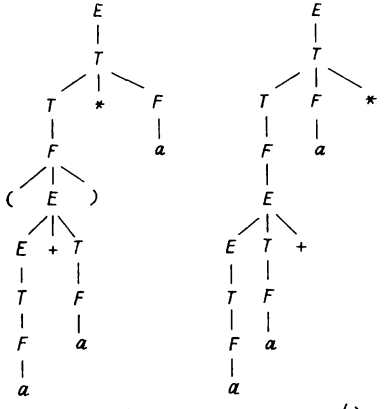
Obě reprezentují jednoduché aritmetické výrazy (se sčítáním a násobením). Gramatika $\mathscr{G}_1$ zadává výrazy v obvyklé infixové

notaci, pravá generuje výrazy v postfixové, tzv. polské notaci. Naším cílem je překládat výrazy v infixové notaci na odpovídající výrazy v postfixové notaci. Tzn. např. výraz $(a + a) * a$ má být přeložen na výraz $aa + a *$. Překlad uskutečníme pomocí stromového automatu, který jako vstup bude dostávat derivační stromy podle gramatiky $\mathscr{G}_1$ a jako výstup bude vydávat jim odpovídající derivační stromy podle $\mathscr{G}_2$. Takovým automatem je např. deterministický automat s jediným stavem q_0 a pravidly

$$
\begin{aligned}
q_0(E[x_1x_2x_3]) &\to E[q_0(x_1)\, q_0(x_3) +], \\
q_0(E[x_1]) &\to E[q_0(x_1)], \\
q_0(T[x_1x_2x_3]) &\to T[q_0(x_1)\, q_0(x_3) *], \\
q_0(T[x_1]) &\to T[q_0(x_1)], \\
q_0(F[x_1x_2x_3]) &\to F[q_0(x_2)], \\
q_0(F[x_1]) &\to F(a).
\end{aligned}
$$

Čtenář se může sám přesvědčit, že např. strom z obr. 50a) odpovídající výrazu $(a + a) * a$ je automatem transformován na strom z obr. 50b) odpovídající výrazu $aa + a *$.

V dalším příkladu využijeme možnosti stromových automatů převádět stromy na lineární řetězy.



a) b) Obr. 50

Příklad 7.3.16. Popíšeme stromový automat, který bude realizovat překlad přiřazovacích příkazů z jazyka vyššího typu do asembleru. Příkazy, jimiž se budeme zabývat, budou tvaru

$$\langle \text{identifikátor} \rangle := \langle \text{term} \rangle ,$$

kde ⟨term⟩ označuje aritmetický výraz dovolující sčítání a násobení. Pro jednoduchost budeme předpokládat, že příkazy uvedeného typu budou zadávány ve formě derivačních stromů gramatiky

$$P \to a := T,$$

$$T \to S \mid N \mid a,$$

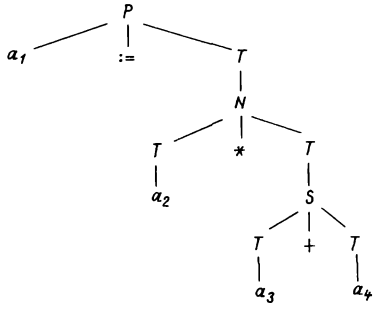
$$S \to T + T,$$

$$N \to T * T,$$

kde P, T, S, N jsou neterminály, P počáteční symbol.

V naší gramatice označuje terminál a libovolný identifikátor. (Čtenář si sám může jako cvičení sestrojit automat pro podobné přiřazovací příkazy, v nichž termy budou reprezentovány derivačními stromy podle gramatiky $\mathscr{G}_1$ z předchozího příkladu.)

Tak např. přiřazovacímu příkazu $X := Z * (X + Y)$ odpovídá derivační strom z obr. 51.



Obr. 51

V konkrétních aplikacích by jednotlivým výskytům terminálu a odpovídaly směrníky ukazující na adresu příslušné konstanty či proměnné. Proto jsou na obr. 51 jednotlivé výskyty a očíslovány. To také usnadní orientaci ve výpočtu.

Jazyk, do něhož se budou přiřazovací příkazy překládat, bude mít příkazy LOAD *a*, STORE *a*, ADD, MULT. Tyto příkazy budou řídit činnost střádače organizovaného ve formě zásobníku. Příkaz LOAD *a* uloží hodnotu z adresy *a* na vrchol zásobníku, příkaz STORE *a* uloží vrchní hodnotu zásobníku na adresu *a*, příkaz ADD (resp. MULT) vyjme dvě vrchní hodnoty ze zásobníku, sečte (resp. vynásobí) je a výsledek uloží na vrchol zásobníku. Překlad přiřazovacího příkazu zadaného stromem popsaného typu lze nyní snadno reprezentovat deterministickým stromovým automatem s jediným stavem a pravidly

$$
\begin{aligned}
&q_0(P[x_1x_2x_3]) \to \text{LOAD } q_0(x_3);\ \text{STORE } q_0(x_1),\\
&q_0(T[x_1]) \to q_0(x_1),\\
&q_0(S[x_1x_2x_3]) \to q_0(x_1);\ \text{LOAD } q_0(x_3);\ \text{ADD},\\
&q_0(N[x_1x_2x_3]) \to q_0(x_1);\ \text{LOAD } q_0(x_3);\ \text{MULT},\\
&q_0(a) \to a.
\end{aligned}
$$

Strom z obr. 51 je popsaným automatem zpracováván takto (pro lepší přehlednost vznikající řetěz členíme vertikálně):

$$
\begin{aligned}
&q_0(P[a_1 := T[N[T[a_2] * T[S[T[a_3] + T[a_4]]]]]]) \vdash\\
&\vdash \left\{\begin{array}{l}\text{LOAD } q_0(T[N[T[a_2] * T[S[T[a_3] + T[a_4]]]]]);\\ \text{STORE } q_0(a_1)\end{array}\right\} \vdash\\
&\vdash \left\{\begin{array}{l}\text{LOAD } q_0(N[T[a_2] * T[S[T[a_3] + T[a_4]]]]);\\ \text{STORE } a_1\end{array}\right\} \vdash\\
&\vdash \left\{\begin{array}{l}\text{LOAD } q_0(T[a_2]);\\ \text{LOAD } q_0(T[S[T[a_3] + T[a_4]]]);\\ \text{MULT};\\ \text{STORE } a_1\end{array}\right\} \vdash\\
&\vdash \left\{\begin{array}{l}\text{LOAD } q_0(a_2);\\ \text{LOAD } q_0(S[T[a_3] + T[a_4]]);\\ \text{MULT};\\ \text{STORE } a_1\end{array}\right\} \vdash
\end{aligned}
$$

$$\vdash \left\{\begin{array}{l} \text{LOAD } a_2; \\ \text{LOAD } q_0(T[a_3]); \\ \text{LOAD } q_0(T[a_4]); \\ \text{ADD}; \\ \text{MULT}; \\ \text{STORE } a_1 \end{array}\right\} \vdash$$

$$\vdash \left\{\begin{array}{l} \text{LOAD } a_2; \\ \text{LOAD } q_0(a_3); \\ \text{LOAD } q_0(a_4); \\ \text{ADD}; \\ \text{MULT}; \\ \text{STORE } a_1 \end{array}\right\} \vdash$$

$$\vdash \left\{\begin{array}{l} \text{LOAD } a_2; \\ \text{LOAD } a_3; \\ \text{LOAD } a_4; \\ \text{ADD}; \\ \text{MULT}; \\ \text{STORE } a_1 . \end{array}\right.$$

Uvedený automat provede překlad správně za předpokladu, že předložený strom je skutečně derivačním stromem podle příslušné gramatiky. Automat je ovšem možno zdokonalit tak, že překlad předloženého stromu dokončí pouze tehdy, jestliže předložený strom je skutečně derivačním stromem podle gramatiky $\mathscr{G}$. Jeho pravidla pak budou vypadat takto:

$q_P(P[x_1x_2x_3]) \to \text{LOAD } q_T(x_3);\ q_{:=}(x_2)\ \text{STORE } q_1(x_1)$,

$q_{:=}(:=) \to e$,

$q_a(a) \to a$,

$q_T(T[x_1]) \to q_S(x_1)$,

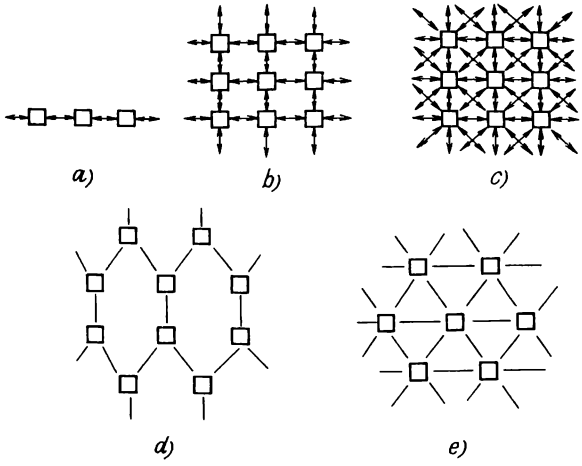
$q_T(T[x_1]) \to q_N(x_1)$,

$q_S(S[x_1x_2x_3]) \to q_T(x_1);\ q_+(x_2)\ \text{LOAD } q_T(x_3);\ \text{ADD}$,

$q_N(N[x_1x_2x_3]) \to q_T(x_1)$; $q_*(x_2)$ LOAD $q_T(x_3)$; MULT,
$q_+(+) \to e$,
$q_*(*) \to e$.

7.4. Celulární automaty

V našem výběru příkladů automatů, které se vymykají obvyklé struktuře (vstupní páska, konečná řídicí jednotka, paměť) se dostáváme k tzv. celulárním automatům.



Obr. 52

Celulární automat si můžeme představit jako nekonečně mnoho exemplářů jistého konečného automatu propojených určitým uniformním způsobem. Příklady takového propojení jsou na obr. 52.

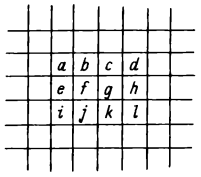
Jednotlivé konečné automaty, tvořící „tkáň" celulárního automatu, budeme nazývat *buňkami*. Každá buňka je spojena s několika buňkami, které nazýváme *okolím* dané *buňky*. Předpokládáme, že všechny buňky v celulárním automatu pracují synchronně, tzn. stavy jednotlivých buněk se mění ve všech buňkách zároveň v jednotlivých okamžicích diskrétní časové škály. Stav buňky v následujícím okamžiku je určen přítomným stavem buňky a přítomným stavem buněk jejího okolí. Např. u celu-

lárních automatů se strukturou znázorněnou na obr. 52 se na určení stavu buňky podílí (po řadě) 3, 5, 9, 4 a 7 buněk.

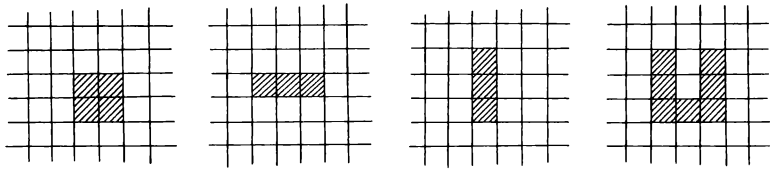
Příklad 7.4.1. Populárním příkladem celulárního automatu je tzv. *Conwayova hra „Life"*, představující jednoduchý model chování kolonií mikroorganismů. Buňky tohoto automatu jsou uspořádány jako na obr. 52c) a jednotlivé buňky nabývají pouze dvou stavů: stavu 1 (živá buňka) a stavu 0 (mrtvá buňka). Přechodová funkce jednotlivých buněk je dána takto:

1. Jestliže je buňka ve stavu 0 a počet „sousedů" ve stavu 1 je
a) 0 až 2, potom buňka zůstane ve stavu 0;
b) 3 až 4, potom přejde do stavu 1;
c) 5 až 8, potom zůstane ve stavu 0.
2. Jestliže je buňka ve stavu 1 a počet „sousedů" ve stavu 1 je
a) 0 až 1, potom buňka přejde do stavu 0;
b) 2 až 4, potom zůstane ve stavu 1;
c) 5 až 8, potom přejde do stavu 0.

Tento automat lze reprezentovat rovinou s čtvercovou sítí. Potom např. následující stav buňky označené na obr. 53 písmenem *f* se určí na základě stavu buňky *f* samotné a jejích „sousedů" *a*, *b*, *c*, *e*, *g*, *i*, *j*, *k*. Stav buňky *g* pak obdobně z *g*, *b*, *c*, *d*, *f*, *h*, *j*, *k*, *l*. Například automat na obr. 54a) (černě jsou označeny buňky ve stavu 1) zůstane v dalším okamžiku beze změny,



Obr. 53



Obr. 54

automat na obr. 54b) přejde na automat z obr. 54c) a ten opět přejde na automat znázorněný na obr. 54b), jak se můžete sami přesvědčit.

Zkuste si sestrojit několik po sobě následujících stavů automatu z obr. 54d).

Podobně jako v předchozím příkladu mají i obecně buňky celulárních automatů jeden význačný stav, tzv. *klidový stav*. Jestliže je buňka i celé její okolí v klidovém stavu, potom i v následujícím taktu zůstane v klidovém stavu. Proto tedy celulární automat, jehož všechny buňky jsou v klidovém stavu, zůstává trvale beze změny.

Jestliže však je část buněk v jiném než klidovém stavu, může se postupně celkový obraz celulárního automatu měnit. Přitom se mohou měnit nejen stavy *aktivních buněk* (tj. buněk mimo klidový stav), ale mohou se některé buňky dostat z klidového stavu a naopak některé aktivní buňky mohou do klidového stavu přejít.

Buňky v jiném než klidovém stavu tvoří tzv. *konfiguraci*. Konfigurace je určena nejen tím, které buňky do ní patří, ale i tím, v kterém stavu se každá z nich nachází.

Z vlastností klidového stavu a z předpokladu, že každá buňka závisí jen na konečném okolí, plyne, že od konečné konfigurace přejde celulární automat vždy jen ke konečné konfiguraci.

Bývá zvykem studovat homogenní a izotropní celulární automaty, tj. automaty, u nichž vývoj konfigurace je invariantní vůči posunutí a otočení. Podrobněji: jsou-li K_1, K_1' dvě konfigurace takové, že K_1' vzniká z K_1 jistou transformací t složenou z posunutí a otočení, a jestliže K_1 v jednom kroku přejde na K_2 a K_1' na K_2', potom také K_2' vzniká z K_2 transformací t.

Téměř výlučná pozornost bývá věnována konečným konfiguracím. Obvykle je zvolen jeden pevný celulární automat, který představuje jakési „životní prostředí", uvnitř něhož se vyvíjejí jednotlivé konečné konfigurace, jejichž vlastnosti jsou zkoumány.

Oblastí aplikací se celulární automaty dosti odlišují od automatů probíraných v předchozích partiích této knihy. Své uplatnění nacházejí zejména při modelování biologických systémů,

jejichž elementární stavební prvky reagují v závislosti jen na velmi úzkém okolí. [Viz např. článek [26] o modelování určitých částí mozkových tkání nebo kolonií bakterií (článek [14]).]

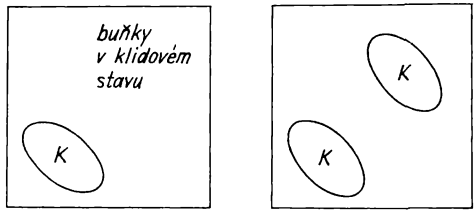
Samotný pojem celulárního automatu byl zaveden von Neumannem v souvislosti s jinou zajímavou otázkou – sebereprodukcí automatů. Tuto otázku rozebereme poněkud podrobněji.

Každodenní zkušenost učí, že stroje produkují výrobky jednodušší, než jsou ony samy. Nedovedeme si dost dobře představit, jak by měl vypadat stroj, který by byl schopen vyrobit kopii sebe sama. Proto vzbudil zaslouženou pozornost nejen von Neumannův výsledek, podle kterého existence takového stroje je z matematického hlediska v principu možná, ale i jeho popis takového stroje.

Projděme si základní myšlenkové postupy, které vedly k návrhu sebereprodukujícího automatu.

Vhodným nástrojem jsou zde právě celulární automaty, které mohou hrát roli životního prostředí, uvnitř kterého automaty utvořené z „živých" buněk (tzn. konfigurace) mohou transformovat své okolí a vytvářet objekty stejné povahy, jako jsou ony samy. Je proto možné vyjít od pojmu sebereprodukující konfigurace. Jestliže celulární automat zahájí činnost s jistou konfigurací [viz obr. 55a)], která se postupně vyvíjí tak, že po nějaké době přejde do konfigurace, jež je zdvojením původní konfigurace [viz obr. 55b)], potom původní konfiguraci nazveme *sebereprodukující*.

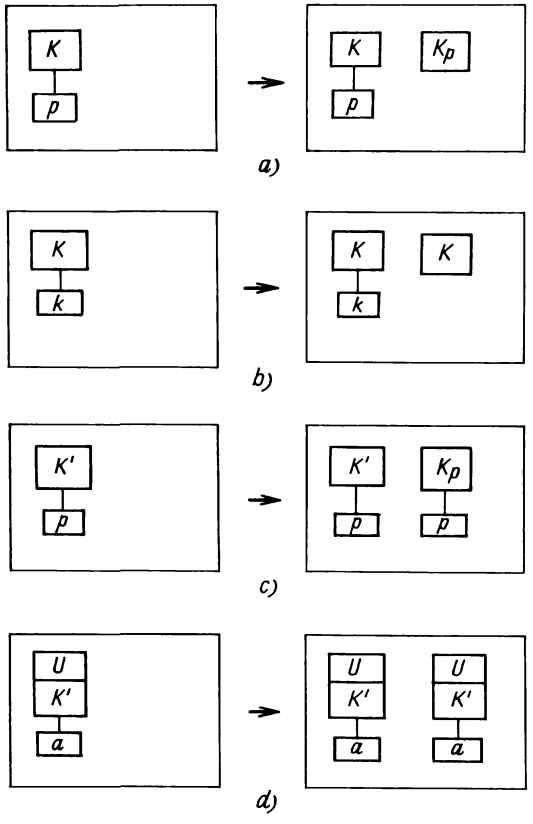
Sestrojit nějaký celulární automat, na němž by bylo možné vytvořit sebereprodukující konfiguraci, není problém. Sami snadno



a) b) Obr. 55

navrhnete celulární automat se strukturou např. podle obr. 52a), který by z konfigurace sestávající z jediné aktivní buňky ve stavu q přešel po několika okamžicích do konfigurace tvořené dvěma aktivními buňkami ve stavu q oddělenými jednou buňkou v klidovém stavu. Buňky tohoto automatu by navíc mohly mít velmi komplikovanou strukturu automatu o mnoha tisících stavech, takže bychom zdánlivě získali sebereprodukci libovolně složitých automatů.

To by ovšem byl pouze formální trik, který by se vyhnul hlavnímu problému, o který jde v pracech zabývajících se sebereprodukcí automatu. Problém spočívá ve vytvoření komplikovaného



Obr. 56

automatu (s možnostmi univerzálního počítače) nadaného schopností sebereprodukce, a to pomocí buněk o nevelkém počtu vnitřních stavů a malém okolí. V práci [29] byl takový automat vytvořen z buněk o 29 stavech a čtyřprvkovém okolí – struktura automatu podle obr. 52b), a v knize [6] automat s toutéž strukturou a okolím a dokonce s buňkami o 8 stavech.

Přitom se postupuje následujícím způsobem. Základním úkolem je najít konfiguraci K, která je schopna podle plánu zkonstruovat každou konfiguraci z jisté třídy $\mathscr{K}$. To znamená: najít jisté kódování (opět pomocí konfigurací) konfigurací z $\mathscr{K}$ a konfiguraci K tak, aby po spojení K s libovolným kódem p vytvořila konfiguraci K_p, jejímž kódem je p [viz obr. 56a)].

Podstatné je vytvořit konfiguraci K, která by konstruovala tak širokou třídu $\mathscr{K}$, aby zahrnovala i samotnou konfiguraci K. Jestliže pak připojíme ke K její vlastní kód k, vytvoří svou vlastní kopii [viz obr. 56b)].

Jelikož $\mathscr{K}$ potřebuje k vytvoření své kopie ještě informaci k, která se do nově vytvořené K nepřenese, není zatím sebereprodukce dokonalá. Proto nestačí nalézt uvedenou konfiguraci K, ale je třeba sestrojit K', která pro všechny konfigurace z $\mathscr{K}$ na základě jejich plánu p vytvoří K_p a připojí k němu kopii p [viz obr. 56c)].

Jestliže se K' podaří navrhnout tak, aby patřila do $\mathscr{K}$, jsme hotovi. Stačí ke K' připojit její vlastní plán k' a K' vytvoří kopii K' i s plánem k'.

Je dokonce možné rozšířit K', pokud to nenaruší činnost K', a vzniklá konfigurace stále ještě patří do $\mathscr{K}$. Bývá proto např. možné rozšířit K' o další úsek U, který je schopen pracovat jako univerzální Turingův stroj a neruší se vzájemně s K'. Jestliže takto vzniklá konfigurace A má kód a, dostaneme po jeho připojení k A sebereprodukující univerzální stroj [viz obr. 56d)].

LITERATURA

[1] *Ábrahám, S.:* Some Questions of Phrase Structure Grammars. Computational Linguistics, *4*, 1965.

[2] *Aho, A. V. – Hopcroft, J. E. – Ullman, J. D.:* The Design and Analysis of Computer Algorithms. Reading (Mass.), Addison-Wesley Publ. Comp., Inc., 1976.

[3] *Aho, A. V. – Ullman, J. D.:* The Theory of Parsing, Translation and Compiling, Vol. 1. Parsing. Englewood Cliffs (N. J.), Prentice Hall 1972.

[4] *Aho, A. V. – Ullman, J. D.:* The Theory of Parsing, Translation and Compiling, Vol. 2. Compiling. Englewood Cliffs (N. J.), Prentice Hall 1973.

[5] *Bar-Hillel, Y. – Gaifman, H. – Shamir, E.:* On Categorial and Phrase Structure Grammars. Bull. Res. Council Israel, *9*, 1960.

[6] *Codd, E. F.:* Cellular Automata. New York, Academic Press 1968.

[7] *Cook, S. A.:* Characterization of Pushdown Machines in Terms of Time-Bounded Computers. JACM, *18*, 1971, č. 1.

[8] *Engelfriet, J.:* Two-Way Automata and Checking Automata, Mathematical Centre Tracts 108. – „Foundations of Computer Science III", Amsterdam 1979.

[9] *Engelfriet, J.:* Some Open Questions and Recent Results on Tree Transducers and Tree Languages. In: Book, R. V. (ed.): Formal Language Theory, Perspectives and Open Problems. New York, Academic Press 1980.

[10] *Fischer, M. J.:* Grammars with Macro-Like Productions. [Dizertační práce.] Cambridge (Mass.) 1968. – Harvard University.

[11] *Frijters, D. – Lindenmayer, A.:* A Model for the Growth and Flowering of Aster Novae-Angliae on the Basis of Table $\langle 1, 0 \rangle$ L-Systems. In: Rozenberg, G. – Salomaa, A. (eds.): L-Systems. Lecture Notes in Comput. Sci. *15*. Berlin, Springer-Verlag 1974.

[12] *Gardner, M.:* The Fantastic Combinations of John Conway's New Solitaire Game „Life". Scientific American, *10*, 1970.

[13] *Ginsburg, S.:* Algebraic and Automata-Theoretic Properties of Formal Languages. Amsterdam, North-Holland Publ. Comp. 1975.

[14] *Goodmann, E. – Weinberg, R. – Laing, R.:* A Cell Space Embedding of Simulated Living Cells. Bio-Medical Computing, *2*, 1971.

[15] *Gray, J. N. – Harrison, M. A. – Ibarra, G. H.:* Two-Way Pushdown Automata. Inf. and Control, *11*, 1967.

[16] *Greibach, S. A.:* Checking Automata and One-Way Stack Languages. Journal of Computer and System Sciences, *3*, 1969.

[17] *Harrison, M. A.:* Introduction to Formal Language Theory. Reading (Mass.), Addison-Wesley Publ. Comp., Inc., 1978.

[18] *Hartmanis, J. – Stearns, R. E.:* Algebraic Theory of Sequential Machines. Englewood Cliffs (N. J.), Prentice-Hall, Inc., 1966.

[19] *Hennie, F.:* Finite-State Models for Logical Machines. London, John Wiley 1969.

[20] *Hopcroft, J. E. – Ullman, J. D.:* Formal Languages and Their Relation to Automata. Reading (Mass.), Addison-Wesley. Publ. Comp., Inc., 1969. (Slovenský překlad: Formálne jazyky a automaty. Bratislava, Alfa 1978.)

[21] *Hopcroft, J. E. – Ullman, J. D.:* Introduction to Automata Theory, Languages and Computation. Reading (Mass.), Addison-Wesley. Publ. Comp. Inc., 1979.

[22] *Chytil, M.:* Teorie automatů a formálních jazyků. [Skripta.] Praha, SPN 1978.

[23] *Ibarra, O.:* Simple Matrix Grammars. Inf. and Control, *17*, 1970.

[24] *Knuth, D. E.:* On the Translation of Languages From Left to Right. Inf. and Control, *8*, 1965.

[25] *Lindenmayer, A.:* Mathematical Models for Cellular Interactions in Development, Parts I–II. Journal of Theoretical Biology, *18*, 1968.

[26] *Mortimer, J. A.:* A Cellular Model for Mammalian Cerebellar Cortex. Tech. Report LCG-107, University of Michigan 1970.

[27] *Rogers, H.:* Theory of Recursive Functions and Effective Computability. New York, McGraw-Hill Book Comp. 1967.

[28] *Salomaa, A.:* Formal Languages. New York, Academic Press, Inc., 1973.

[29] *Thatcher, J. W.:* Universality in the von Neumann Cellular Model. Tech. Report 03105-30-T, ORA, University of Michigan 1964.

[30] *Turing, A. M.:* On Computable Numbers, with an Application to the Enscheidungs Problem. Proc. London Math. Soc., Serie 2, *42*, 1936.

RNDr. Michal Chytil, CSc.

AUTOMATY A GRAMATIKY

DT 681.3
681.3.04

Vydalo SNTL – Nakladatelství technické literatury, n. p.,
Spálená 51, 113 02 Praha 1
v roce 1984
jako svou 9687. publikaci
Redakce teoretické literatury
Odpovědná redaktorka RNDr. Blanka Kutinová, CSc.
Obálku navrhl Miroslav Houska
Technický redaktor Jan Šulc
Vytiskla Polygrafia, n. p., Svobodova 1, Praha 2
336 stran, 56 obrázků, 19 tabulek
Typové číslo L11-E1-IV-41f/11 878. Vydání první
Náklad 3200 výtisků. 19,89 AA, 20,40 VA

03/2
Cena brožovaného výtisku Kčs 22,–
505/21, 826

Publikace je určena zejména systémovým programátorům, posluchačům a učitelům vysokých škol z oborů vychovávajících odborníky zaměřené na používání výpočetní techniky. Může posloužit i učitelům a studentům gymnázií s výukou programování

04-012-84