Studijní opora k předmětu Teoretická informatika
(speciálně v oblasti teorie jazyků a automatů)

Petr Jančar

25. září 2003

Obsah

Kapitola 1

Úvod, cíle kursu, literatura

Přednáška 1

Teorie jazyků a automatů patří ke klasickým partiím teoretické informatiky a je ve větší či menší míře součástí studijních plánů informatiky na všech vysokých školách.

Zde předkládaný materiál je zamýšlen jako studijní opora pro úvodní kurs v dané oblasti. Materiál vznikl přepracováním a doplněním pracovního textu, jenž původně sloužil jen jako doprovodný materiál k přednáškám. Nynější text by měl umožňovat (resp. více podporovat) i samostatné či distanční studium. Speciálně byly do textu doplněny úkoly, o jejichž vyřešení je studující na příslušně označených místech v textu žádán.

Poznámka. Na jaře 2003 převedli diplomanti L. Jamrozová a L. Koblovský text do interaktivnější elektronické podoby – využitím TeXovského balíku maker "Elearn" J. Dvorského a dodáním ilustrativních animací k vybraným pojmům a postupům.

Text je členěn logicky do kapitol, z nichž každá obsahuje stručně specifikované studijní cíle. Jelikož materiál je samozřejmě i nadále možno použít jako podporu k řádnému kursu s přednáškami (tedy nejen k distančnímu studiu), je v textu vyznačeno i orientační rozdělení na jednotlivé přednášky (v délce 2 x 45 minut). Počítá se, že každá přednáška je doplněna (rovněž dvouhodinovým) cvičením, na které se ovšem studenti předem samostatně připravují. Z těchto orientačních údajů je možné odhadnout časovou náročnost studia jednotlivých kapitol či jejich částí – jde ovšem čistě o orientační odhad, jelikož u každého jednotlivce bude čas potřebný k důkladnému zvládnutí látky velmi individuální.

Velmi přínosná by samozřejmě byla snaha studujícího prostudovat probírané partie také v některé z doporučených (či jiných) monografií. Ty najdete, stejně jako další informace ke kursu, přes odkaz na webovské stránce

http://www.cs.vsb.cz/jancar
(či přímo http://www.cs.vsb.cz/jancar/TJAA/tjaa.htm)

Autor bude vděčný za jakékoli připomínky k předkládanému materiálu (stejně jako obecně ke kursu), sdělené např. e-mailem na adresu Petr.Jancar@vsb.cz

---

Teorie jazyků a automatů patří k základním (a dnes již klasickým) partiím teoretické informatiky, partiím, jejichž vznik a vývoj byl a je úzce svázán s potřebami praxe při vývoji software, hardware a obecně při modelování systémů.

Náš kurs tvoří určitý celek s kursem *Vyčíslitelnost a složitost* (plánovaným v dalším semestru), dohromady by se mohly nazývat *Základy teorie výpočtů* (Theory of Computation). Motivovat teorii výpočtů lze přirozenými otázkami typu:
- jak srovnat kvalitu (rychlost) různých algoritmů řešících tentýž problém (úkol) ?
- jak lze porovnávat (klasifikovat) problémy podle jejich (vnitřní) složitosti ?
- jak charakterizovat problémy, které jsou a které nejsou algoritmicky řešitelné, tj. které lze a které nelze řešit algoritmy (speciálně "rychlými", neboli prakticky použitelnými, algoritmy).

Při zpřesňování těchto a podobných otázek, a při hledání odpovědí, nutně potřebujeme (abstraktní) modely počítače (tj. toho, kdo provádí výpočty). Z více důvodů je vhodné při našem zkoumání začít velmi jednoduchým, ale fundamentálním modelem, a sice tzv. *konečnými (tj. konečně stavovými) automaty*. Konečné automaty (pojem byl formalizován ve 40. letech 20. století), lze chápat nejen jako nejzákladnější model v oblasti počítačů, ale ve všech oblastech, kde jde o modelování systémů, procesů, organismů apod., u nichž lze vyčlenit konečně mnoho stavů a popsat způsob, jak se aktuální stav mění prováděním určitých akcí (např. přijímáním vnějších impulsů). (Jako jednoduchý ilustrující příklad nám může posloužit model ovladače dveří v supermarketu znázorněný na obr. 2.1, o němž pojednáme níže.) Velmi běžná "výpočetní" aplikace, u níž je v pozadí konečný automat, je *hledání vzorků v textu*. Takové hledání asi nejčastěji používáme v textových editorech a při vyhledávání na Internetu; speciální případ také představuje např. *lexikální analýza* v překladačích. Při vyhledávání informací v počítačových systémech

jste již jistě narazili na nějakou variantu *regulárních výrazů*, umožňujících specifikovat celé třídy vzorků. Regulárními výrazy a jejich vztahem ke konečným automatům se rovněž budeme zabývat.

Po seznámení se s konečnými automaty a regulárními výrazy budeme pokračovat silnějším modelem – tzv. *zásobníkovými automaty*; o ty se opírají algoritmy *syntaktické analýzy* při překladu (programovacích) jazyků, tedy algoritmy, které např. určí, zda vámi napsaný program v Pascalu (C-čku) je správně pascalsky (c-čkovsky).

Zmínili jsme pojem *jazyk* – obecně budeme mít na mysli tzv. formální jazyk; jazyky přirozené (mluvené) či jazyky programovací jsou speciálními případy. Na naše modely se v prvé řadě budeme dívat jako na *rozpoznávače jazyků*, tj. zařízení, které zpracují vstupní posloupnost písmen (symbolů) a rozhodnou, zda tato posloupnost je (správně utvořenou) větou příslušného jazyka.

Poznámka. Ve skutečnosti je zřejmě důležitější (a přirozenější) pojem *funkce* realizované daným automatem – automat k zadanému vstupu ("spočítá" a) vydá jistý výstup, tedy realizuje určitou (vstupně-výstupní) funkci. Rozpoznávač jazyka je pouze speciálním případem, kdy výstupem je 1 či 0 (ANO či NE). (Později se ještě o tomto problému zmíníme.)

S pojmem *jazyk* se nám přirozeně pojí pojem *gramatika*. Speciálně se budeme věnovat tzv. *bezkontextovým gramatikám*, s nimiž jste se již přinejmenším implicitně setkali u definic syntaxe programovacích jazyků (tj. pravidel konstrukce programů); ukážeme mj., že bezkontextové gramatiky generují právě ty jazyky, jež jsou rozpoznávány zásobníkovými automaty.

Seznámíme se také s jedním z univerzálních (nejobecnějších) modelů počítačů (resp. rozpoznávačů jazyků) – s tzv. *Turingovými stroji*. K tomuto modelu se také odkazuje zmíněný kurs o vyčíslitelnosti a složitosti.

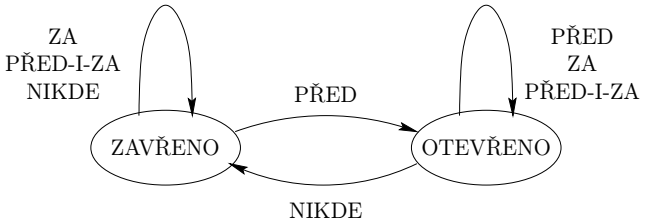
Ještě poznamenejme, že účelem kursu není popis konkrétních "reálných" (větších) aplikací studovaných (teoretických) pojmů; účelem je základní seznámení se s těmito pojmy a s příslušnými obecnými výsledky a metodami. Jejich zvládnutí je nezbytným základem pro porozumění i oněm reálným aplikacím a pro jejich návrh. Jako pěkný příklad relativně nedávné aplikace může sloužit použití konečných automatů s vahami pro reprezentaci složitých funkcí (na reálném oboru) a jejich využití při reprezentaci, transformaci a kompresi obrazové informace (viz např. kapitolu v J. Gruska: Foundations of Computing, Intern. Thomson Computer Press, 1997).

Kapitola 2

Konečné automaty, regulární jazyky

Konečný automat je systém (či model systému), který může nabývat konečně mnoho (obvykle ne "příliš mnoho") stavů. Daný stav se mění na základě vnějšího podnětu (možných podnětů také není "příliš mnoho") s tím, že pro daný stav a daný podnět je jednoznačně určeno, jaký stav bude následující (tj. do jakého stavu systém přejde). Konkrétní konečný automat se často zadává diagramem, který také nazýváme *stavový diagram* či *graf automatu*. Jiná možnost zadání je *tabulkou*, která je sice poněkud suchopárnější, ale např. je vhodnější pro počítačové zpracování.

*Příklad.*Diagram na obr. 2.1 je popisem jednoduchého automatu řídícího vstupní dveře do supermarketu (dveře nejsou posuvné, ale otvírají se dovnitř). Automat může být ve dvou stavech (Zavřeno, Otevřeno) a podnětem (např. snímaným v pravidelných krátkých intervalech) je informace, na které z podložek (před dveřmi, za dveřmi) se někdo nachází. Kromě (pro naše oko přehledného) diagramu lze tutéž informaci sdělit tabulkou na obr. 2.2.



Obrázek 2.1:

	PŘED	ZA	PŘED-I-ZA	NIKDE
ZAVŘENO	OTEVŘENO	ZAVŘENO	ZAVŘENO	ZAVŘENO
OTEVŘENO	OTEVŘENO	OTEVŘENO	OTEVŘENO	ZAVŘENO

Obrázek 2.2:

Úkol 1 *Popište slovně nějaký jednoduchý automat na mince, který je schopen vydat čaj nebo kávu dle volby, a pak jej modelujte stavovým diagramem (grafem automatu).*

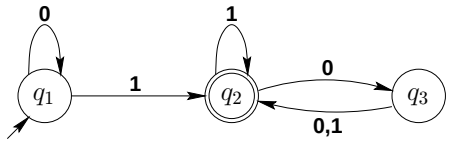
Na základě uvedeného jednoduchého příkladu máme již představu, co to konečný automat je. Formalizujme nyní tento pojem v jazyce matematiky. K čemu je to dobré ? Pro další zkoumání potřebujeme přesnou (a jednoznačnou) definici a také stručné a přehledné značení. (U takto jednoduchého pojmu bychom se bez formalizace snad ještě obešli, u složitějších pojmů později už těžko.)

Definice 2.1 *Konečný automat, zkráceně KA, A je pětice (tzn., že je dán pěticí parametrů)* $A = (Q, \Sigma, \delta, q_0, F)$*, kde* Q *je konečná neprázdná množina stavů,* Σ *je konečná neprázdná množina zvaná (vstupní)* abeceda*,* $\delta : Q \times \Sigma \to Q$ *je* přechodová funkce*,* $q_0 \in Q$ *je počáteční (iniciální) stav a* $F \subseteq Q$ *je množina přijímajících (koncových) stavů.*

Všimněme si, že uvedená definice vyžaduje určení počátečního stavu a tzv. přijímajících (nebo též koncových) stavů – o těch dosud nebyla řeč. Vymezení těchto stavů je důležité v případě, o který se zvlášť budeme zajímat, tj. v případě, kdy konečný automat hraje roli rozpoznávače jazyka. Zatím řekneme jen neformálně: jazyk rozpoznávaný daným konečným automatem je množina posloupností symbolů (prvků abecedy), které automat přijímá (akceptuje), tj. těch posloupností, které automat převedou z počátečního do některého z přijímajících stavů.

Jak už víme, (malý) konečný automat lze často přehledně reprezentovat *grafem automatu*; dohodněme se, že počáteční stav budeme označovat malou vstupní šipkou, přijímající stavy pak dvojitým kolečkem.

*Poznámka.*V řeči teorie grafů bychom přesněji měli mluvit o ohodnoceném *multigrafu* – mezi dvěma vrcholy může vést více hran, ohodnocených prvky abecedy. Běžně ovšem takové hrany na obrázku znázorňujeme hranou jedinou, na níž uvedeme všechna příslušná ohodnocení; na obr. 2.3 to ilustruje hrana vedoucí ze stavu q_3 do stavu q_2 (zastupuje hranu s ohodnocením 0 i hranu s ohodnocením 1).



přijímá:
1101, 010101, . . .

nepřijímá:
0110, 0010, . . .

Obrázek 2.3:

Úkol 2 *Chápejme obr. 2.3 jako popis konečného automatu* $A = (Q, \Sigma, \delta, q_0, F)$. *Vypište* přímým výčtem *hodnoty všech parametrů automatu A. Pak porovnejte s obr. 2.7. Na obrázku je počáteční stav* q_1 *rovnou zapsán jako čtvrtý parametr místo* q_0*; mohli bychom to samozřejmě také řešit zápisem* $q_0 = q_1$.

Obr. 2.3 již čtenáři jistě dal představu o tom, co to znamená, že daný konečný automat přijímá danou posloupnost symbolů. Je ovšem opět užitečné a žádoucí pojmy zformalizovat. Začneme pojmem jazyk a souvisejícími definicemi; zároveň zavedeme i značení používané později.

Definice 2.2 *Abeceda je konečná neprázdná množina* symbolů *(písmen, znaků, signálů apod.). Často ji označujeme znakem* Σ*, její prvky pak znaky* a, b, c *(s případnými indexy apod.).*

Slovo v abecedě Σ *je libovolná konečná posloupnost* $a_1, a_2, \ldots, a_n$ *prvků* Σ. *Píšeme ji obvykle bez čárek –* $a_1a_2 \ldots a_n$. *Slova označujeme symboly* u, v, w *apod. Přirozeným způsobem definujeme délku slova, označujeme* $|u|$*; pro* $u = a_1a_2 \ldots a_n$ *je* $|u| = n$. *Symbolem* $|u|_a$ *označujeme počet výskytů symbolu a ve slově u.*

Speciální roli hraje prázdné slovo (s délkou 0*); označujeme jej* ε. Zřetězení slov $u = a_1a_2 \ldots a_n$, $v = b_1b_2 \ldots b_m$ *označujeme* $u \cdot v$, *stručněji* uv, *a definujeme* $uv = a_1a_2 \ldots a_nb_1b_2 \ldots b_m$. *Výrazem* u^n *označujeme n-násobné zřetězení slova u; je tedy* $u^0 = \varepsilon$, $u^1 = u$, $u^2 = uu$, $u^3 = uuu$ *atd.*

Slovo u je podslovem slova v, jestliže v lze psát $v = w_1uw_2$ *pro nějaká* w_1, w_2 *(jinak řečeno: jestliže existují slova* w_1, w_2 *taková, že* $v = w_1uw_2$*). Slovo u je prefixem (sufixem) slova v, jestliže* $v = uw$ $(v = wu)$ *pro nějaké w.*

Σ^* *označuje množinu všech slov a* Σ^+ *množinu všech neprázdných slov v abecedě* Σ. *(Je tedy* $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$*.)*

Formální jazyk, stručně jazyk, nad abecedou Σ *je libovolná množina slov v abecedě* Σ. *Jazyky obvykle označujeme L (s indexy); pro jazyk L nad abecedou* Σ *je tedy* $L \subseteq \Sigma^*$.

*Příklad.*Slova v abecedě $\Sigma = \{a, b\}$ jsou např. $\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba$.

Je užitečné si uvědomit, že slova v každé abecedě lze přirozeně uspořádat (a systematicky generovat) jak jsme naznačili v příkladu: v prvé řadě podle délky (kratší předchází delšímu) a v rámci stejné délky abecedně (lexikograficky) – což předpokládá, že máme uspořádány prvky abecedy. Např. pro $\Sigma = \{0, 1\}$ (s abecedním uspořádaním $0 < 1$) lze prvky Σ^* generovat v pořadí $\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots$. Příslušné uspořádání budeme označovat $<_L$(např. $11 <_L 001$).

*Vsuvka.*Připomeňme i obecnou definici uspořádání: relace ρ na množině A je *(částečným) uspořádáním*, jestliže je reflexivní ($\forall x \in A : x\rho x$), tranzitivní ($\forall x, y, z \in A : (x\rho y \wedge y\rho z) \Longrightarrow x\rho z$)) a antisymetrická ($\forall x, y \in A : (x\rho y \wedge y\rho x) \Longrightarrow x = y$)). Hovoříme-li o *uspořádání*, obvykle myslíme úplné (neboli lineární) uspořádání, což znamená navíc $\forall x, y \in A : (x\rho y \vee y\rho x)$ (tedy neexistují vzájemně nesrovnatelné prvky).
Pro relace uspořádání se obvykle používají symboly jako $\leq$ a jemu podobné. Použijeme-li pak symbolu $<$, myslíme obvykle podrelaci "ostrého" (tj. irreflexivního) uspořádání definovanou takto: $x < y \iff_{df} x \leq y \wedge x \neq y$.

Říkáme-li pouze "jazyk", rozumíme tím, že příslušná abeceda je buď zřejmá z kontextu nebo na ní nezáleží.

*Poznámka.*Byť v praktických případech má abeceda např. desítky prvků, v našich příkladech bude abeceda často (jen) dvouprvková (většinou $\{a, b\}$, $\{0, 1\}$). Uvědomme si, že to není zásadní omezení, jelikož písmena víceprvkové abecedy lze přirozeně zakódovat řetězci dvouprvkové abecedy. (Jak dlouhé řetězce byste použili např. při kódování 256-ti prvkové abecedy ?)

*Poznámka.*Uvědomme si, že operace zřetězení slov je asociativní (tzn. $(u \cdot v) \cdot w = u \cdot (v \cdot w)$); proto je např. zápis $u \cdot v \cdot w$ jednoznačný i bez uvedení závorek.

Úkol 3 *Vypište všechna slova v abecedě* $\{a, b\}$*, která mají délku 3.*
Napište explicitně slovo u (posloupnost písmen), které je určeno výrazem $(ab)^3 \cdot ba \cdot (bba)^2$ *(slovo u je tedy výsledkem provedení operací uvedených ve výrazu).*
Vypište všechna slova délky 2, které jsou podslovy slova 00010 *(v abecedě* $\{0, 1\}$*).*
Vypište všechny prefixy (sufixy) slova 0010. *(Je jich pět !)*

Všimněme si ještě, že s jazyky—jakožto s množinami—je možné provádět množinové operace; např. výsledky operací $L_1 \cup L_2$ (sjednocení), $L_1 \cap L_2$ (průnik), $L_1 - L_2$ (množinový rozdíl), $\overline{L}$ (doplněk – rozumí se pro příslušnou abecedu Σ, tj. $\overline{L} = \Sigma^* - L$) jsou opět jazyky. Později zavedeme i speciální jazykové operace.

Úkol 4 *Uvažujme jazyky*
$L_1 = \{w \in \{a,b\} \mid w$ *obsahuje sudý počet výskytů symbolu* $a\}$,
$L_2 = \{w \in \{a,b\} \mid w$ *začíná a končí stejným symbolem* $\}$.
Vypište prvních šest slov (rozumí se v uspořádání $<_L$*) postupně pro jazyky* $L_1 \cup L_2$, $L_1 \cap L_2$, $L_1 - L_2$, $\overline{L}$.

Pokračujeme přesnou definicí jazyka rozpoznávaného konečným automatem; příslušné jazyky budeme nazývat regulární.

Definice 2.3 *Přechodovou funkci automatu* $A = (Q, \Sigma, \delta, q_0, F)$ *zobecníme na funkci*
$\delta^* : Q \times \Sigma^* \to Q$ *touto induktivní definicí:*

1. $\delta^*(q, \varepsilon) = q$,
2. $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$.

Úkol 5 *Automat A na obrázku 2.4 nejprve zadejte výčtem parametrů,* $A = (Q, \Sigma, \delta, q_0, F)$.
Z induktivní definice δ^* *pak detailně odvoďte, čemu se rovná* $\delta^*(2, \texttt{bab})$.

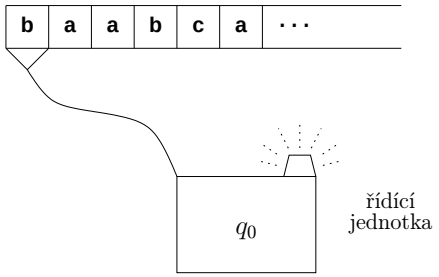
Důkladně si formu a obsah induktivní definice promyslete. Zároveň se přesvědčte, že nemůže dojít k nedorozumění, když v dalším budeme místo δ^* psát jen δ, je totiž $\delta^*(q, a) = \delta(q, a)$. ($\delta^*$ je tedy rozšířením δ na větší definiční obor; z kontextu bude vždy zřejmé, odkazujeme-li se na δ v užším nebo širším smyslu.)

Definice 2.4 *Slovo* $w \in \Sigma^*$ *je* přijímáno *automatem A, jestliže* $\delta(q_0, w) \in F$. Jazykem rozpoznávaným (přijímaným) automatem A *rozumíme jazyk* $L(A) = \{w \mid$ *slovo* w *je přijímáno* $A\}$.
Jazyk L *je* regulární *(tj. rozpoznatelný konečným automatem), jestliže existuje KA A tž.* $L(A) = L$.

Úkol 6 *Ověřte si, že uvedená definice jazyka rozpoznávaného KA skutečně zachycuje (formalizuje) předcházející neformální vysvětlení. Dále zkuste formálně nadefinovat graf* G_A *automatu A a definnujte jazyk* $L(A)$ *pomocí (pojmů teorie grafů na) grafu* G_A.

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

Obrázek 2.4:



Obrázek 2.5:

Konečný automat si lze představovat různě. Pro naše účely je zřejmě nejvhodnější představa znázorněná na obr. 2.5. *Řídicí jednotka*, což je "skříňka" nabývající konečně mnoha (vnitřních) stavů (řekněme několika-bitová paměť), je *čtecí hlavou* spojena se (vstupní) *páskou*, na níž je zapsáno slovo – zleva doprava jsou v jednotlivých buňkách pásky uložena písmena daného slova.

Na začátku je řídicí jednotka v počátečním stavu a hlava je připojena k nejlevějšímu políčku pásky. Činnost automatu, zvaná *výpočet*, pak probíhá v *krocích*: v každém kroku je přečten symbol (hlava se po přečtení posune o jedno políčko doprava) a řídicí jednotka se nastaví do stavu určeného aktuálním stavem a přečteným symbolem (přechodová funkce je v řídicí jednotce "zadrátovaná").

Je možné si také představit, že na řídicí jednotce je "světélko" signalizující navenek, zda aktuální stav je či není přijímající (čili "zvenku" rozlišujeme u řídicí jednotky jen dva stavy – přijímá/nepřijímá).

Uvedená prezentace konečného automatu vede k další variantě definice přijímaných slov. Uvedeme ji teď hlavně proto, že pozdější definice pro další modely budou pak jen přímočarým zobecněním. Všimněte si také, že přitom budeme explicitně definovat pojem *výpočtu* automatu.

*Vsuvka.*Připomeňme nejdříve některé elementární pojmy teorie množin a algebry. *Kartézským součinem* $M \times N$ množin M, N rozumíme množinu dvojic $M \times N = \{ (a,b) \mid a \in M, b \in N \}$. *(Kartézské) mocniny* množiny M lze definovat takto: $M^1 = M, M^2 = M \times M, M^3 = M \times (M \times M), \ldots, M^{i+1} = M \times M^i, \ldots$. *$n$-ární relace na množině* M je podmnožina M^n. Nejčastěji uvažované relace jsou 2-ární (neboli binární), proto pojmem *relace ρ na množině* M rozumíme *binární relaci*, tedy $\rho \subseteq M \times M$; přitom často píšeme $x\rho y$ místo $(x,y) \in \rho$. Relace ρ na množině M je *reflexivní* právě tehdy, když $\forall x \in M : x\rho x$; ρ je *tranzitivní* právě tedy, když $\forall x, y, z \in M : (x\rho y \wedge y\rho z) \Longrightarrow x\rho z$. *Reflexivním a tranzitivním uzávěrem* relace ρ rozumíme nejmenší relaci ρ' (nejmenší ve smyslu množinové inkluze), která obsahuje ρ a je přitom reflexivní i tranzitivní.

Definice 2.5 *Konfigurací konečného automatu $A = (Q, \Sigma, \delta, q_0, F)$ rozumíme dvojici (q,w), kde $q \in Q$ a $w \in \Sigma^*$ (q představuje aktuální stav a w slovo, které zbývá přečíst – připomeňme, že čtecí hlava se pohybuje jen doprava).*

Na množině všech konfigurací automatu A definujeme relaci $\vdash_A$ takto: $(q, aw) \vdash_A (q', w)$ právě když $\delta(q,a) = q'$ (rozumí se $a \in \Sigma$, $w \in \Sigma^*$). Zápis $K_1 \vdash_A K_2$ čteme např. "konfigurace K_1 bezprostředně (tj. v jednom kroku) vede ke konfiguraci K_2", "konfigurace K_2 bezprostředně následuje za K_1" apod.

Výpočtem *automatu A, začínajícím v konfiguraci K, rozumíme posloupnost konfigurací $K_0, K_1, K_2, \ldots, K_n$, kde $K_0 = K$ a $K_i \vdash_A K_{i+1}$ pro $i = 0, 1, \ldots, n-1$ (takový výpočet má délku n, tj. sestává z n kroků).*

Výpočet $K_0, K_1, K_2, \ldots, K_n$ je přijímajícím výpočtem *pro slovo w, jestliže $K_0 = (q_0, w)$ a $K_n = (q, \varepsilon)$ pro nějaký $q \in F$.*

Slovo $w \in \Sigma^*$ je přijímáno *KA A, jestliže existuje přijímající výpočet pro slovo w.*

Alternativně lze také definovat:

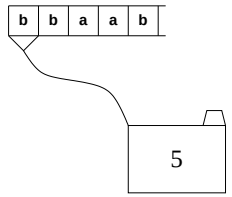
Relace $\vdash_A^*$ je reflexivním a tranzitivním uzávěrem relace $\vdash_A$. $K_1 \vdash_A^* K_2$ pak čteme např. takto: "konfigurace K_1 vede ke K_2", "K_1 odvodí K_2", "K_2 je následníkem K_1" apod.

Slovo $w \in \Sigma^*$ je přijímáno *KA A, jestliže $(q_0, w) \vdash^* (q, \varepsilon)$ pro nějaký přijímající stav $q \in F$. (Místo $\vdash_A^*$ píšeme jen $\vdash^*$, jestliže automat A, k němuž se daná relace vztahuje, je zřejmý z kontextu.)*

*Poznámka.*Všimněme si, že zápisy $\delta(q,w) = q'$ a $(q,w) \vdash^* (q', \varepsilon)$ mají stejný význam. Šlo by se také např. dohodnout, že $(q,w) \vdash^* (q', \varepsilon)$ budeme zapisovat elegantněji $q \xrightarrow{w} q'$ apod. Čtenář je vyzýván, ať si další definice a tvrzení používající notaci s δ přeformuluje i dalšími uvedenými způsoby. Tím mj. vyzdvihujeme obecný fakt, že totéž sdělení (tutéž sémantiku, tj. tentýž význam pojmů, tvrzení apod.) lze zachytit různými syntaktickými způsoby (značeními), z nichž každý může mít své výhody a nevýhody.

Úkol 7 *Pro automat A na obrázku 2.4*

- simulujte krok po kroku výpočet na slově `bbaab` *(zapište příslušnou posloupnost konfigurací; první, tedy $(5, bbaab)$, je zachycena na obrázku 2.6).*



Obrázek 2.6: Počáteční konfigurace automatu A

- vypište všechna slova délky ≤ 3 v abecedě $\{$`a`, `b`$\}$ a zjistěte, která z nich patří do jazyka $L(A)$.

- zakreslete graf (stavový diagram) automatu A

- charakterizujte co nejjednodušeji jazyk $L(A)$ (vlastností slov do něj náležejících)

Lze snadno nahlédnout, že pro jazyk $L(A)$ hrají roli jen ty stavy automatu A, které jsou dosažitelné (z počátečního stavu):

Definice 2.6 Stav q *automatu $(Q, \Sigma, \delta, q_0, F)$ je* dosažitelný*, jestliže existuje $w \in \Sigma^*$ tž. $\delta(q_0, w) = q$.*

Jako procvičení formy induktivní definice můžeme dosažitelnost uvést také takto:
Množina *dosažitelných stavů* automatu $(Q, \Sigma, \delta, q_0, F)$ je *nejmenší* množina

$A = (Q, \Sigma, \delta, q_1, F)$, kde

$$\begin{array}{rcll} Q & = & \{q_1, q_2, q_3\} & \delta(q_1, 0) = q_1, \quad \delta(q_1, 1) = q_2, \\ \Sigma & = & \{0, 1\} & \delta(q_2, 0) = q_3, \quad \delta(q_2, 1) = q_2, \\ F & = & \{q_2\} & \delta(q_3, 0) = q_2, \quad \delta(q_3, 1) = q_2 \end{array}$$

Obrázek 2.7:

$K \subseteq Q$ splňující tyto dvě podmínky: 1/ $q_0 \in K$, 2/ jestliže $q \in K$ a $q' = \delta(q, a)$ pro nějaké $a \in \Sigma$, potom $q' \in K$.

Tato definice je de facto přímým návodem k sestavení algoritmu, který zjistí všechny dosažitelné stavy. Připomeňme, že konečný automat můžeme zadat přímým výčtem všech parametrů (např. automat z obr. 2.3 je takto popsán na obr. 2.7), ale také grafem (stavovým diagramem) či tabulkou (do té je rovněž nutno přidat označení počátečního a přijímacích stavů).

Úkol 8 *Zformulujte algoritmus, který pro daný KA, reprezentovaný grafem, označí všechny dosažitelné stavy; algoritmus pak zformulujte pro případ reprezentace tabulkou.*
Aplikujte na automat z obrázku 2.4.

Jak jsme už zmínili, odstraníme-li nedosažitelné stavy (a příslušně tedy omezíme definiční obor přechodové funkce), rozpoznávaný jazyk se nemění. Máme tedy

Tvrzení 2.0.1 *Ke každému KA A lze zkonstruvat KA A', v němž každý stav je dosažitelný a $L(A') = L(A)$.*

Přednáška 2

Z tvrzení 2.0.1 snadno nahlédneme, že

Věta 2.0.2 *Existuje algoritmus, který pro zadaný KA A rozhodne, zda $L(A)$ je neprázdný.*

(Jak vypadá ten algoritmus ?)
Pro čtenáře teď bude jistě snadné ukázat i dva následující fakty:

Tvrzení 2.0.3 *Pro konečný automat A s n stavy je $L(A)$ neprázdný právě tehdy, když existuje $w \in L(A)$ délky menší než n ($|w| < n$).*

Tvrzení 2.0.4 *Pro konečný automat A s n stavy je $L(A)$ nekonečný právě tehdy, když existuje $w \in L(A)$ splňující $n \leq |w| < 2n$.*

Úkol 9 *Načrtněte* rychlý *algoritmus, rozhodující pro daný A, zda $L(A)$ je nekonečný.*

Doplnění.

Víme, že i nekonečné množiny mohou mít různou mohutnost. (Množinu přirozených čísel nazýváme spočetnou, množinu reálných čísel nespočetnou.) V této souvislosti je užitečné si uvědomit následující fakty.

Z uspořádání $<_L$je zřejmé, že

Tvrzení 2.0.5 *Pro (konečnou neprázdnou) abecedu Σ je Σ^* nekonečná spočetná množina (tj. existuje bijekce, neboli vzájemně jednoznačné zobrazení, mezi Σ^* a množinou $\mathcal{N}$ všech přirozených čísel).*

Poznámka. Všimněme si, že ze spočetnosti Σ^* (pro lib. abecedu Σ) plyne např. spočetnost množiny všech racionálních čísel. (Proč ?)

Z obecné Cantorovy věty, která říká, že mohutnost množiny M je ostře menší než mohutnost množiny všech jejích podmnožin $\mathcal{P}(M)$, plyne

Tvrzení 2.0.6 *Jazyků nad (neprázdnou) abecedou Σ je nespočetně mnoho.*

Např. už z těchto úvah o mohutnostech množin plyne, že ne každý jazyk je regulární – konečných automatů je totiž spočetně mnoho ! (Proč ?)

Normovaný tvar konečného automatu

V dalším občas využijeme (speciálně pak u zkušebních testů !) jistý přirozeně definovaný normovaný tvar konečného automatu, v němž jsou všechny stavy dosažitelné. Lze si to představit tak, že každému stavu q přiřadíme nejkratší slovo (přesněji: nejmenší slovo vzhledem k uspořádání $<_L$, pomocí něhož je q dosažitelný (z počátečního stavu), a stavy pak uspořádáme podle těchto přiřazených slov, přičemž je označujeme čísly $1, 2, \ldots, n$. Přesněji můžeme definovat takto:

Definice 2.7 *Máme-li dán konečný automat $A = (Q, \Sigma, \delta, q_0, F)$ bez nedosažitelných stavů a uspořádání (prvků) abecedy Σ (které indukuje uspořádání $<_L$ na Σ^*), pak řekneme, že A je v normovaném tvaru, jestliže*

- $Q = \{1, 2, \ldots, n\}$ *(pro nějaké* $n \geq 1$*),*
- 1 *je počáteční stav,*
- *označíme-li pro každý stav* $i \in \{1, 2, \ldots, n\}$ *symbolem* u_i *nejmenší slovo v uspořádání* $<_L$*, pro něž* $\delta(1, u_i) = i$*, pak pro* $i < j$ *platí* $u_i <_L u_j$*.*

Úkol 10 *Rozmyslete si jednoduchý algoritmus, který každý KA převede (přejmenováním stavů) do normovaného tvaru.*
Návod: počáteční stav označíme 1. Dále, např. v případě abecedy $\{a, b\}$*, zjistíme stav* q*, do něhož automat přejde ze stavu 1 symbolem* a*; když* q *není označen (jako 1), označíme jej 2. Pak zjistíme stav* q*, do něhož automat přejde ze stavu 1 symbolem* b*; když* q *není dosud označen, označíme jej nejmenším dosud nepoužitým číslem. Takto jsme "vyřídili" stav 1, pokračujeme "vyřízením" 2 atd.*
Aplikujte na automat z obrázku 2.4; všimněte si přitom, že se takto automaticky vymezíme (označíme) právě všechny dosažitelné stavy

Úkol 11 *Navrhněte konečný automat s abecedou* $\{\texttt{0}, \texttt{1}, \texttt{2}, \langle\text{RESET}\rangle\}$*, který přijímá ta slova, kde součet symbolů* `0`*,* `1`*,* `2` *(braných numericky) za posledním symbolem* $\langle\text{RESET}\rangle$ *(či od začátku, není-li* $\langle\text{RESET}\rangle$*) dává při dělení 3 zbytek 1.*

Úkol 12 *Navrhněte konečný automat s abecedou* $\{\texttt{0}, \texttt{1}\}$ *přijímající právě ta slova, v nichž je sudý počet (výskytů) symbolů* `0` *a každý symbol* `1` *je bezprostředně následován symbolem* `0`*.*
Dokažte správnost vašeho návrhu, tedy vysvětlete, proč vámi navržený automat splňuje daný požadavek.

Úkol 13 *Navrhněte konečný automat přijímající právě ta slova v abecedě* $\{a, b\}$*, u nichž prefix délky 2 se rovná sufixu délky 2 (slova mají délku alespoň 2).*
Snažte se, aby tento automat měl minimální počet stavů – můžete se přitom také pokusit vysvětlit, proč vámi navržený počet stavů nelze zmenšit.

Úkol 14 *Pro konečný automat* $A = (Q, \Sigma, \delta, q_0, F)$*,* $\Sigma = \{a, b\}$*, uveďte induktivní definici funkce* $R_a : \mathcal{P}(Q) \to \mathcal{P}(Q)$*, pro niž, neformálně řečeno, platí toto:*

stav q *patří do* $R_a(K)$ *právě tehdy, když se do* q *lze dostat z nějakého* $q' \in K$ *jen pomocí* a*-šipek (tedy nějakým slovem tvaru* a^i*).*

Úkol 15 *Navrhněte konečný automat rozpoznávající jazyk* $L_1 = \{\, w \in \{a, b\}^* \mid w$ *obsahuje podslovo* $aba \,\}$ *a konečný automat rozpoznávající jazyk*

$L_2 = \{\, w \in \{a, b\}^* \mid |w|_b \bmod 2 = 0 \,\}$ *(v* L_2 *jsou tedy právě slova obsahující sudý počet* b*-ček). Pak zkonstuujte automat rozpoznávající jazyk* $L_1 \cap L_2$*; zkuste přitom vhodně využít automaty zkonstruované pro jazyky* L_1*,* L_2*.*

Úkol 16 *Na vybraných zkonstruovaných automatech si procvičte převod do normovaného tvaru.*

Kapitola 3

Návrh konečných automatů. Regulární operace.

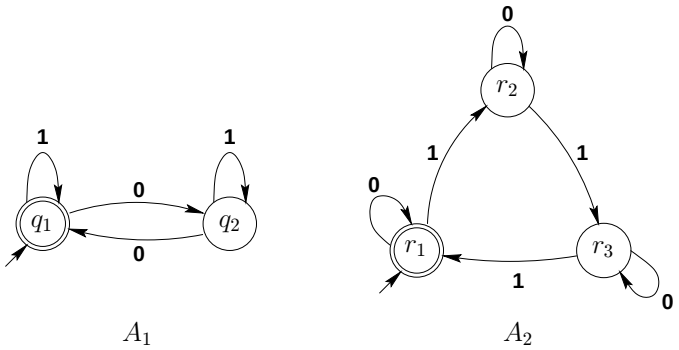
Návrh konečného automatu, který bude rozpoznávat zadaný jazyk, je tvůrčí činnost, vlastně jakési jednoduché programování, a proto nelze podat "mechanický" návod, jak automaty vytvářet. Stačí ovšem důkladně promyslet pár příkladů, aby člověk s programátorskou zkušeností (a tedy schopný "vžít se do role konečného automatu") tuto činnost zvládl.

Mnoho věcí se přitom zmechanizovat (zalgoritmizovat) dá. Např. existují algoritmy, které z automatů pro "jednodušší" jazyky vytvářejí automaty pro jazyky vzniklé operacemi nad oněmi (jednoduššími) jazyky.

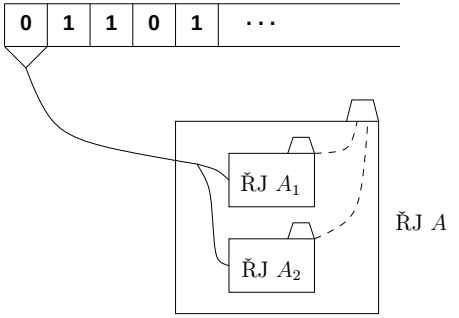
Představme si např., že chceme sestrojit automat, který má rozpoznávat jazyk, jenž je sjednocením dvou regulárních jazyků. Pro konkrétnost např. jazyk sestávající ze slov v abecedě $\{0,1\}$, v nichž počet nul je dělitelný dvěma nebo počet jedniček je dělitelný třemi. Tedy jazyk $L = L_1 \cup L_2$, kde $L_1 = \{w \in \{0,1\}^* \mid |w|_0 \text{ je dělitelné } 2\}$ a $L_2 = \{w \in \{0,1\}^* \mid |w|_1 \text{ je dělitelné } 3\}$. (Označením $|w|_a$ značíme počet výskytů symbolu a ve slově w.) Na obr. 3.1 jsou znázorněny automaty rozpoznávající jazyky L_1 a L_2.

Jak rozpoznáme, zda dané slovo patří do $L(A_1) \cup L(A_2)$? Prostě je necháme zpracovat oběma automatům A_1, A_2 a podíváme se, zda alespoň jeden z nich skončil v přijímajícím stavu. Ovšem tuto naši činnost může očividně provádět i jistý konečný automat A – viz obr. 3.2. Rozmyslete si, co jsou stavy automatu A, co je to počáteční a koncový stav, a jak vypadá přechodová funkce.

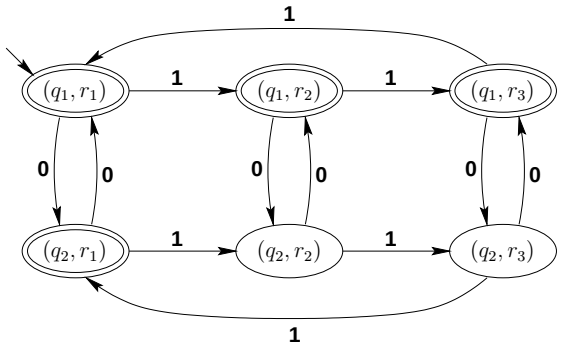
Pak se podívejte na obr. 3.3, znázorňující A pro náš konkrétní případ, a přesvědčte se, že to, co jste pochopili a co jste si odvodili, je přesně to,



Obrázek 3.1:



Obrázek 3.2:



Obrázek 3.3:

co je díky matematické notaci přesně a velmi stručně zachyceno v důkazu následující věty.

Věta 3.0.7 *Jestliže jazyky $L_1, L_2 \subseteq \Sigma^*$ jsou regulární, pak také jazyk $L_1 \cup L_2$ je regulární.*

Důkaz. Nechť $L_1 = L(A_1)$, $L_2 = L(A_2)$ pro konečné automaty $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$.
Definujme automat $A = (Q, \Sigma, \delta, q_0, F)$ tž.

- $Q = Q_1 \times Q_2$,
- $\delta(\,(q_1, q_2), a\,) = (\,\delta_1(q_1, a), \delta_2(q_2, a)\,)$ pro vš. $q_1 \in Q_1$, $q_2 \in Q_2$, $a \in \Sigma$,
- $q_0 = (q_{01}, q_{02})$,
- $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$.

Je očividné (exaktně lze ukázat např. indukcí podle délky w), že pro lib. $q_1 \in Q_1$, $q_2 \in Q_2$ a $w \in \Sigma^*$ je $\delta(\,(q_1, q_2), w\,) = (\,\delta_1(q_1, w), \delta_2(q_2, w)\,)$.
Z toho snadno plyne, že $L(A) = L_1 \cup L_2$.
Konec Důkazu

Úkol 17 *Mějme konečné automaty bez specifikace počátečních a přijímajících stavů $A_1 = (Q_1, \Sigma, \delta_1)$, $A_2 = (Q_2, \Sigma, \delta_2)$.*

Definujme $A = (Q, \Sigma, \delta)$ tž. $Q = Q_1 \times Q_2$ a $\delta(\,(q_1, q_2), a\,) = (\,\delta_1(q_1, a), \delta_2(q_2, a)\,)$ pro vš. $q_1 \in Q_1$, $q_2 \in Q_2$, $a \in \Sigma$.
Ukažte, že pro lib. $q_1 \in Q_1$, $q_2 \in Q_2$ a $w \in \Sigma^*$ platí

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

Použijte indukci podle délky slova w.

Na základě (důkazu) věty 3.0.7 pro sjednocení teď ukažte analogickou větu pro průnik:

Věta 3.0.8 *Jestliže jazyky $L_1, L_2 \subseteq \Sigma^*$ jsou regulární, pak také jazyk $L_1 \cap L_2$ je regulární.*

(Nápověda: $F = (F_1 \times F_2)$)
*Poznámka.*Je velmi vhodné si uvědomit *konstruktivnost* našich tvrzení. Např. věta 3.0.7 (věta 3.0.8) říká jen to, že sjednocení (průnik) dvou regulárních jazyků je také regulární jazyk. Dokázali jsme však více: existuje *algoritmus*, který ke konečným automatům rozpoznávajícím L_1, L_2 zkonstruuje automat rozpoznávající $L_1 \cup L_2$ ($L_1 \cap L_2$). Podobně tomu bude u dalších vět v tomto kursu, i když se o tom třeba nebudeme explicitně zmiňovat.

Úkol 18 *Automat pro průnik požadovaný v úkolu 15 sestrojte podle obecného návodu z (důkazu) předchozí věty. (Porovnejte se svou předchozí konstrukcí.)*

Zvláštní roli (zmíněnou ještě později) hrají tzv. regulární operace. Vedle (množinové operace) *sjednocení* k nim patří další dvě (jazykové) operace – *zřetězení* a *iterace*. (Průnik mezi regulární operace nezařazujeme. Důvody vysvitnou později.)

Definice 3.1 *Regulárními operacemi s jazyky nazýváme operaci sjednocení, $L_1 \cup L_2 = \{w \mid w \in L_1$ nebo $w \in L_2\}$, zřetězení, $L_1 \cdot L_2 = \{uv \mid u \in L_1, v \in L_2\}$, a iterace, $L^* = \bigcup_{n=0}^{\infty} L^n$, kde L^n je definováno induktivně: $L^0 = \{\varepsilon\}$, $L^{n+1} = L \cdot L^n$.*

Předpokládáme, že čtenáři už nebude dělat potíže pochopit zaváděné pojmy pouze z jejich matematické definice. Odvodí si tak např., že do *zřetězení* dvou jazyků patří právě každé takové slovo, které vznikne, když za nějaké slovo z prvního jazyka postavíme nějaké slovo z druhého jazyka (zřetězíme je). Jinými slovy: takové slovo se dá rozdělit na dvě části, z nichž první patří

do prvního a druhá do druhého jazyka. Mj. si všimneme $\emptyset \cdot L = L \cdot \emptyset = \emptyset$, $\{\varepsilon\} \cdot L = L \cdot \{\varepsilon\} = L$.

Rovněž vidíme, že do *iterace* jazyka L patří právě každé takové slovo, jež lze rozdělit na několik částí, přičemž každá z nich patří do L – tj. vznikne zřetězením několika (konečně mnoha) slov z L. Definice nám také říká, že prázdné slovo patří do iterace vždy (vznikne "zřetězením nula slov z L").

Poznámka. Formálně lze L^* definovat také např. takto: $L^* = \{\, w \mid \text{ex. } n \geq 0$ a slova $u_1, u_2, \ldots, u_n \in L$ tak, že $w = u_1 u_2 \ldots u_n \,\}$.

Všimněme si např., že $\emptyset^* = \{\varepsilon\}$, $(L^*)^* = L^*$ apod. Rovněž si uvědomme, že naše dříve zavedené značení Σ^* je v souladu s nyní uvedenou definicí iterace.

Úkol 19 *Procvičte si definici zřetězení a iterace jazyků na malých příkladech a pak dokažte či vyvraťte obecnou platnost následujících vztahů:*

- $L_1 \cdot L_2 = L_2 \cdot L_1$
- $L_1(L_2 \cup L_3) = L_1 L_2 \cup L_1 L_3$
- $(L_1 \cup L_2)^* = L_1^*(L_2 \cdot L_1^*)^*$
- $(L_1 \cap L_2)^* = L_1^* \cap L_2^*$

Kapitola 4

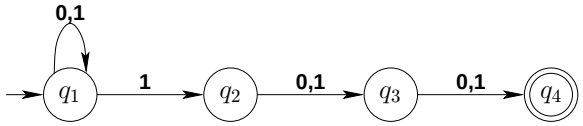
Nedeterministické konečné automaty a jejich vztah k deterministickým.

Uvažujme nyní obdobu věty 3.0.7 pro zřetězení jazyků. Jistě nás napadne přístup: "necháme na první část slova běžet první automat, v přijímajícím stavu pak předáme řízení druhému automatu a ten dočte slovo do konce". Problém je, že obecně nepostačí prostě předat řízení při *prvním* příchodu do přijímajícího stavu (proč ne ?), ale je nutno nechat to jako *možnost při jakémkoli* příchodu do přijímajícího stavu.

Toto chování snadno realizujeme automatem, v němž připustíme *nedeterminismus* – v daném stavu a při daném vstupním symbolu je obecně více možností, do kterého stavu přejít. V našem případě by se nám ještě více hodilo, aby šlo změnit stav, aniž se čte vstupní symbol (při onom "předání řízení"); tato možnost se objeví u ZNKA níže. Nejprve začneme s následující definicí nedeterministického konečného automatu ($\mathcal{P}(Q)$ označuje množinu všech podmnožin množinu Q):

Definice 4.1 *Nedeterministický konečný automat, zkráceně NKA, A je dán pěticí parametrů $A = (Q, \Sigma, \delta, I, F)$, kde Q je konečná neprázdná množina* stavů, Σ *je (konečná neprázdná)* abeceda, $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ *je* přechodová funkce, $I \subseteq Q$ *je množina* počátečních (iniciálních) stavů *a* $F \subseteq Q$ *je množina* přijímajících (koncových) stavů.

Přímočaře lze opět nadefinovat *graf* (též nazývaný stavový diagram) NKA. Příklad takového grafu je na obrázku 4.1 – nadefinujte takový graf obecně pro NKA A ! (Mělo by být zřejmé, že pro NKA $A = (Q, \Sigma, \delta, I, F)$ znázorněný



přijímá např.:
01101

nepřijímá např.:
011010

Obrázek 4.1:

grafem na obrázku 4.1 je např. $\delta(q_1, 1) = \{q_1, q_2\}$, $\delta(q_2, 0) = \{q_3\}$, $\delta(q_4, 1) = \emptyset$ apod.)

Definice tedy připouští, že z daného stavu q pro daný symbol a vychází více "a-šipek" (nebo někdy žádná a-šipka), a také připouští více počátečních stavů.

Obr. 4.1 ukazuje k danému NKA také příklad přijímaného a nepřijímaného slova. Zkuste z toho vyvodit, jak je definováno přijímání slova nedeterministickým konečným automatem, a pak teprve svůj závěr konfrontujte s dále uvedenou definicí.

Přednáška 3

Pro definici *výpočtu* NKA lze takřka doslova použít definici 2.5 – jen místo $\delta(q, a) = q'$ napíšeme $\delta(q, a) \ni q'$ a místo q_0 použijeme např. $r, r \in I$. (Promyslete si podrobně tuto modifikovanou definici, speciálně si promyslete definici přijímání slova v tomto novém kontextu !)

Vidíme tedy, že na rozdíl od KA (užíváme též DKA, když chceme zdůraznit, že máme na mysli standardní, tj. deterministický automat), kde k danému slovu existuje jediný úplný (tj. dokončený, neprodloužitelný) výpočet, u NKA existuje k danému slovu obecně více úplných výpočtů. Rozhodující pro příslušnost slova w k jazyku $L(A)$ (rozpoznávanému NKA A) ovšem je, zda existuje *alespoň jeden* přijímající výpočet pro w.

Přijímání slova a jazyk $L(A)$ lze nadefinovat také takto:

Definice 4.2 Slovo $w \in \Sigma^*$, *tvaru* $w = a_1 a_2 \dots a_n$, *je* přijímáno *NKA* $A = (Q, \Sigma, \delta, I, F)$, *jestliže existují stavy* $q_0, q_1, \dots, q_n$ *takové, že 1/* $q_0 \in I$, *2/* $q_n \in F$, *3/* $\delta(q_{i-1}, a_i) \ni q_i$ *pro vš.* $i = 1, 2, \dots, n$. Jazykem rozpoznávaným (přijímaným) automatem A *rozumíme jazyk* $L(A) = \{w \mid$

	0	1
→1	1,2	1
2	3	-
3	-	4
←4	-	-
→5	5	5,6
6	-	7
7	-	8
8	-	9
←9	9	9

Obrázek 4.2:

slovo w *je přijímáno* $A\}$. Jazyk L *je* rozpoznatelný nedeterministickým konečným automatem *jestliže existuje NKA* A *tž.* $L(A) = L$.

Poznámka. Velmi názorně lze definovat přijímání slova rovněž v termínech grafu NKA. (Jak ?) Všimněme si také explicitně, že dříve uvedený KA lze chápat jako speciální případ NKA (kde $\delta(q, a)$ je vždy jednoprvková množina a také I je jednoprvková množina).
Jak už jsme zmínili dříve„ místo konečný automat (KA) někdy pro zdůraznění říkáme *deterministický* konečný automat (DKA).

Úkol 20 *Charakterizujte (co nejjednodušeji slovně) jazyk, který je přijímán automatem z obrázku 4.1.*
Zkonstruujte deterministický KA, který přijímá tentýž jazyk.

Úkol 21 *Sestrojte co nejjednodušší* nedeterministický *konečný automat, který přijímá právě ta slova v abecedě* $\{0, 1\}$, *jež začínají* 110 *nebo končí* 001 *nebo obsahují* 1111.

Bude nám rovněž užitečná následující definice:

Definice 4.3 *Definiční obor přechodové funkce v NKA* $A = (Q, \Sigma, \delta, I, F)$ *můžeme opět přirozeně zobecnit na* $\delta : Q \times \Sigma^* \longrightarrow \mathcal{P}(Q)$, *resp. ještě obecněji na* $\delta : \mathcal{P}(Q) \times \Sigma^* \longrightarrow \mathcal{P}(Q)$; *a sice induktivní definicí: 1/* $\delta(K, \varepsilon) = K$, *2/* $\delta(K, wa) = \bigcup_{q \in \delta(K,w)} \delta(q, a)$.

Definici si opět řádně promyslete; speciálně si objasněte, co znamená $\delta(K_1, w) = K_2$. Také si všimněme, že $\delta(I, w)$ je množina právě těch stavů, do kterých se NKA může dostat zpracováním (přečtením) slova w (začne-li v některém z počátečních stavů). Můžeme tedy také psát $L(A) = \{w \in \Sigma^* \mid \delta(I, w) \cap F \neq \emptyset\}$.

Úkol 22 *NKA zadaný tabulkou na obrázku fig:NKA-cv4 zadejte grafem. Pak si na něm ilustrujte funkci* $\delta : \mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$ *pro zvolené argumenty.*

Rozšíření funkce δ u NKA (viz definici 4.3) v podstatě demonstruje, že nedeterministický konečný automat lze nahradit deterministickým – byť (obvykle) s podstatně větším počtem stavů:

Věta 4.0.9 *NKA rozpoznávají právě regulární jazyky (a jsou v tomto smyslu ekvivalentní DKA).*

Důkaz. Jelikož každý DKA je de facto speciálním případem NKA, je zřejmé, že každý regulární jazyk je rozpoznáván nějakým NKA.

Pro opačný směr stačí ukázat konstrukci (algoritmus), která pro zadaný NKA $A = (Q, \Sigma, \delta, I, F)$ vydá DKA A_1 tž. $L(A) = L(A_1)$. Stačí definovat $A_1 = (\mathcal{P}(Q), \Sigma, \delta_1, I, F_1)$, kde pro lib. $K \subseteq Q$ (tj. $K \in \mathcal{P}(Q)$) položíme $\delta_1(K, a) = \delta(K, a)$ (zde se odkazujeme k oné rozšířené definici δ) a dále $F_1 = \{K \subseteq Q \mid K \cap F \neq \emptyset\}$.

Důkaz $L(A) = L(A_1)$ je zřejmý:

Pro lib. slovo w platí: $w \in L(A) \iff \delta(I, w) \cap F \neq \emptyset \iff \delta_1(I, w) \in F_1 \iff w \in L(A_1)$.
Konec Důkazu

Chování DKA A_1 je užitečné si představit ve formě "knoflíkové hry" na grafu NKA A: Na začátku leží knoflíky právě na uzlech odpovídajících I. Přijde-li nyní (přečte-li se) symbol a, každý knoflík se posune podle všech možných přechodů (šipek) a – přitom se knoflík může "rozmnožit" či "zmizet". Potom na každém uzlu, na kterém je více než jeden knoflík, ponecháme knoflík jediný – a jsme připraveni přijmout další vstupní symbol. Daný stav (dané rozmístění knoflíků) je přijímající právě když alespoň jeden knoflík leží na uzlu odpovídajícím některému přijímajícímu stavu automatu A.

Všimněte si, že důkaz věty 4.0.9 ukazuje pro NKA s n stavy konstrukci DKA s 2^n stavů (!) Některé stavy u tohoto DKA mohou být ovšem nedosažitelné (tedy některých rozmístění knoflíků nelze v předchozí hře docílit) a omezení se jen na konstrukci dosažitelných stavů může někdy znamenat obrovskou úsporu.

Úkol 23 *Rozmyslete si, jak lze přímočaře konstruovat např. tabulku, která bude obsahovat jen dosažitelné stavy onoho DKA.*
Aplikujte tuto metodu na NKA z obrázku fig:NKA-cv4.

Bohužel jsou případy, kdy všechny stavy DKA dosažitelné jsou a navíc nelze tyto stavy "uspořit" ani jiným způsobem (jedná se o tzv. minimální, nebo též redukovaný, automat, jak o tom budeme hovořit později).

Úkol 24 *Např. k NKA s 5 stavy zadanému následující tabulkou*

	a	b
$\leftrightarrow 1$	2	-
2	3	1,2
3	4	1,3
4	5	1,4
5	1	1,5

se vám nepodaří nalézt ekvivalentní DKA (tj. DKA rozpoznávající tentýž jazyk) s méně než $2^5 = 32$ *stavy. (Zkuste zjistit, proč !)*
Konstrukci přitom snadno zobecníte pro NKA s n *stavy, pro nějž nejmenší ekvivalentní DKA má* 2^n *stavů.*

*Poznámka.*Uvědomme si ovšem, že ke každému NKA s n stavy (např. $n = 1000$) lze příslušný DKA realizovat (např. simulovat na počítači) za použití (pouze) n bitů, byť má daný DKA 2^n stavů. (Jak ?) Čili tento úkol je zvládnutelný, byť by explicitně zkonstruovaný stavový prostor DKA vůbec nebyl uložitelný do paměti počítače ! (Řešit ovšem např. problém dosažitelnosti stavu v DKA při zmíněné n-bitové reprezentaci je pak úplně jiná otázka !)

Vraťme se nyní k úvahám o (nedeterministickém) automatu pro zřetězení dvou regulárních jazyků a připomeňme si, že se nám hodí více jiný druh nedeterminismu – tzv. ε-přechody, díky nimž automat může změnit stav, aniž čte vstupní symbol:

Definice 4.4 *Zobecněný nedeterministický konečný automat (ZNKA)* A *je dán pěticí parametrů* $A = (Q, \Sigma, \delta, I, F)$. *Jediný rozdíl proti NKA spočívá v tom, že přechodová funkce je typu* $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ *a přijímání slova je definováno takto:*

Slovo $w \in \Sigma^*$, *tvaru* $w = a_1 a_2 \dots a_n$, *je* přijímáno *ZNKA* A, *jestliže existují stavy* $q_1^0, q_2^0, \dots, q_{m_0}^0, q_1^1, q_2^1, \dots, q_{m_1}^1, \dots, q_1^n, q_2^n, \dots, q_{m_n}^n$ $(m_i \geq 1)$ *takové, že 1/* $q_1^0 \in I$, *2/* $q_{m_n}^n \in F$, *3/* $\delta(q_{m_{i-1}}^{i-1}, a_i) \ni q_1^i$ *pro vš.* $i = 1, 2, \dots, n$, *4/* $\delta(q_{j-1}^i, \varepsilon) \ni q_j^i$ *pro vš.* $i = 0, 1, 2, \dots, n$, $j = 2, 3, \dots, m_i$.

Jazykem rozpoznávaným (přijímaným) automatem A *rozumíme jazyk* $L(A) = \{w \mid$ *slovo* w *je přijímáno* $A\}$.

	a	b	c	ε
$\rightarrow$1	2	-	-	3
2	1	-	-	-
3	-	4	-	5
4	-	3	-	-
$\leftarrow$5	-	-	6	-
6	-	-	5	-

Obrázek 4.3:

Uvedená definice přijímání slova nevypadá zrovna elegantně, že ? Mělo by být nicméně jasné, co vyjadřuje.

Úkol 25 *ZNKA z obrázku 4.3 zadejte grafem.*
Naformulujte pojem přijímání slova automatem ZNKA v řeči grafů automatů. Charakterizujte jazyk, který je daným automatem přijímán.

Úkol 26 *Upravte definici 2.5 pro případ ZNKA !*

*Poznámka.*Opět tedy máme příklad toho, že jeden a tentýž pojem lze formalizovat více způsoby (které jsou více či méně vhodné).

Uvedeme ještě jeden způsob, hodící se k zobecnění věty 4.0.9:

Mějme dán ZNKA $A = (Q, \Sigma, \delta, I, F)$. Nejprve pro $K \subseteq Q$ definujme $E(K)$ jako množinu stavů dosažitelných z K jen pomocí ε-šipek.

Úkol 27 *Zkuste podat induktivní definici $E(K)$ a pak se teprve podívejte na dále uvedené řešení.*

$E(K)$ je nejmenší množina splňující 1/ $K \subseteq E(K)$, 2/ jestliže $q \in E(K)$ a $q' \in \delta(q, \varepsilon)$, potom $q' \in E(K)$.
Nyní můžeme přechodovou funkci ZNKA rozšířit na $\delta : \mathcal{P}(Q) \times \Sigma^* \longrightarrow \mathcal{P}(Q)$ takto:

1. $\delta(K, \varepsilon) = E(K)$,
2. $\delta(K, wa) = \bigcup_{q \in \delta(K,w)} E(\delta(q, a))$.

Podobně jako u NKA takto dosáhneme, že $\delta(I, w)$ je množina právě těch stavů, do kterých se ZNKA může dostat zpracováním (přečtením) slova w (a tedy $L(A) = \{w \in \Sigma^* \mid \delta(I, w) \cap F \neq \emptyset\}$), a analogicky ("knoflíkovou hrou") jako větu 4.0.9 lze ukázat

Věta 4.0.10 *ZNKA rozpoznávají právě regulární jazyky.*

Všimněme si ještě, že pomocí NKA lze podat jiný, velmi přímočarý, důkaz věty 3.0.7. (Nápověda: definujte vhodně pojem (disjunktního) *sjednocení* (tj. "vedle sebe položení") *dvou NKA*.) Použijeme-li navíc ε-šipek (tedy ZNKA), docílíme navíc snadno toho, aby výsledný NKA měl jediný počáteční stav. (Proč ?)

*Poznámka.*Pro průnik nám ani elegance ε-šipek moc nepomůže. Ovšem uzavřenost třídy regulárních jazyků vůči průniku plyne také např. z uzavřenosti vůči sjednocení a doplňku, o které se zmíníme za chvíli (de Morganova pravidla).

Úkol 28

	a	*b*
$\leftrightarrow$*1*	*2*	*1*
2	*2*	*3*
3	*4*	*3*
4	*5*	*5*
5	*1*	*5*

Zkonstruujte ZNKA rozpoznávající jazyk $L = \{\, uv \mid uav \in L(A) \vee ubv \in L(A) \,\}$, kde A je KA zadaný uvedenou tabulkou. (Slova jazyka L vzniknou ze slov jazyka $L(A)$ vypadnutím jednoho písmene.) (Nápověda: ZNKA bude "obsahovat dvě kopie výchozího KA".) Nakonec alespoň započněte konstrukci DKA pro jazyk L.

Úkol 29 *Navrhněte rámcově realizaci konečného automatu, který vždy pro zadané slovo v abecedě $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ zjistí, zda v něm nějaké podslovo délky tři má alespoň dva (nepřekrývající se) výskyty.*

Kapitola 5

Uzávěrové vlastnosti třídy regulárních jazyků.

Přednáška 4

Uzavřenost třídy regulárních jazyků na zřetězení a iteraci je znázorněna na obr. 5.1 a 5.2, které by měly dostatečně naznačit myšlenku. Následuje formální důkaz.

Věta 5.0.11 *Jestliže jazyky $L_1, L_2 \subseteq \Sigma^*$ jsou regulární, pak také jazyk $L_1 \cdot L_2$ je regulární. Jestliže L je regulární, pak také L^* je regulární.*

Důkaz. Nechť $L_1 = L(A_1)$, $L_2 = L(A_2)$ pro konečné automaty $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$; můžeme předpokládat $Q_1 \cap Q_2 = \emptyset$.

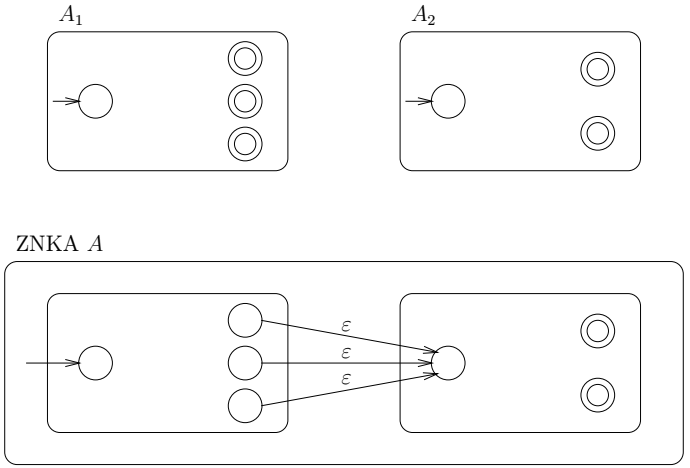
Definujme nyní ZNKA $A = (Q_1 \cup Q_2, \Sigma, \delta, \{q_{01}\}, F_2)$ tak, že $\delta(q, a) = \{\delta_1(q, a)\}$ je-li $q \in Q_1$ a $\delta(q, a) = \{\delta_2(q, a)\}$ je-li $q \in Q_2$; navíc pro každý stav $q \in F_1$ je $\delta(q, \varepsilon) = \{q_{02}\}$ a pro $q \notin F_1$ je $\delta(q, \varepsilon) = \emptyset$.

Je snadné ověřit, že $L(A) = L_1 \cdot L_2$. (Ověřte !)

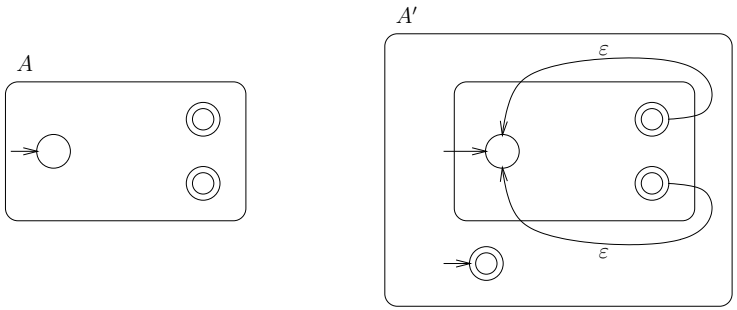
Nechť nyní $L = L(A)$ pro KA $A = (Q, \Sigma, \delta, q_0, F)$.

Definujme ZNKA $A' = (Q \cup \{p\}, \Sigma, \delta', \{q_0, p\}, F \cup \{p\})$, kde $p \notin Q$, $\delta'(q, a) = \{\delta(q, a)\}$ $(a \in \Sigma)$ a pro $q \in F$ je $\delta'(q, \varepsilon) = \{q_0\}$; pro $q \notin F$ je $\delta'(q, \varepsilon) = \emptyset$ a navíc $\delta'(p, a) = \emptyset$ také pro vš. $a \in \Sigma$.

Je snadné ověřit, že $L(A') = L^*$. (Ověřte !)
Konec Důkazu



Obrázek 5.1: $L(A) = L(A_1) \cdot L(A_2)$



Obrázek 5.2: $L(A') = L(A)^*$

Úkol 30 *Uvědomili jste si roli přidaného (izolovaného) počátečního a přijímajícího stavu p v důkazu uzavřenosti na iteraci ?*
Uvažujme pro KA $A = (Q, \Sigma, \delta, q_0, F)$ *konstrukci ZNKA* $A' = (Q, \Sigma, \delta', \{q_0\}, F \cup \{q_0\})$, *kde* $\delta'(q, a) = \{\delta(q, a)\}$ $(a \in \Sigma)$ *a pro* $q \in F$ *je* $\delta'(q, \varepsilon) = \{q_0\}$; *pro* $q \notin F$ *je* $\delta'(q, \varepsilon) = \emptyset$. *Ukažte, že obecně neplatí* $L(A') = L(A)^*$.

Vedle ConceptUsageregulárních operacíregulární operceRegOp (sjednocení, zřetězení, iterace), je množina regulárních jazyků uzavřena i vůči dalším operacím. Uzavřenost vůči průniku jsme již ukázali dříve (věta 3.0.8), snadno ukážeme také uzavřenost na doplněk a vyvodíme uzavřenost na (množinový) rozdíl:

Věta 5.0.12 *Jestliže* L *je regulární, pak také* $\overline{L}$ *je regulární. Jestliže* L_1, L_2 *jsou regulární jazyky, pak také* $L_1 - L_2$ *je regulární.*

Důkaz. Nechť $L = L(A)$, kde A je KA $(Q, \Sigma, \delta, q_0, F)$. Pak $\overline{L}$ je zřejmě rozpoznáván KA $(Q, \Sigma, \delta, q_0, Q-F)$. (Přijímající a nepřijímající stavy byly prohozeny.)
Druhá část tvrzení pak již plyne z toho, že $L_1 - L_2 = L_1 \cap \overline{L_2}$.
Konec Důkazu

Úkol 31 *Uvažujme automaty* A_1, A_2 *zadané tabulkami:*

A_1

	a	b
$\rightarrow q_1$	q_2	q_3
$\leftarrow q_2$	q_2	q_4
$\leftarrow q_3$	q_5	q_3
q_4	q_2	q_4
q_5	q_5	q_3

A_2

	a	b
$\rightarrow r_1$	r_2	r_1
r_2	r_2	r_3
$\leftarrow r_3$	r_2	r_1

Zkonstruujte obecně použitelným algoritmem KA A rozpoznávající jazyk $L(A) = L(A_1) - L(A_2)$. *Poté se snažte jazyk* $L(A)$ *co nejjednodušeji charakterizovat (podmínkou, kterou splňují slova do něj patřící).*

Všimněme si ještě dalších jazykových operací (pojem kvocientu vyžaduje hlubší promyšlení !):

Definice 5.1 Zrcadlový obraz slova $u = a_1 a_2 \dots a_n$ *je* $u^R = a_n a_{n-1} \dots a_1$, zrcadlový obraz jazyka L *je* $L^R = \{u \mid \exists v \in L : u = v^R\}$.

Levý kvocient jazyka L_1 *podle jazyka* L_2 *je jazyk* $L_2 \backslash L_1 = \{ u \mid \exists v \in L_2 : vu \in L_1 \}$.
Pravý kvocient jazyka L_1 *podle jazyka* L_2 *je jazyk* $L_1 / L_2 = \{ u \mid \exists v \in L_2 : uv \in L_1 \}$.

Úkol 32 *Rozmyslete si, proč obecně platí:*

- $(L^R)^R = L$
- $(L_1 \cdot L_2)^R = L_2^R \cdot L_1^R$
- $L/\emptyset = \emptyset$
- $L/\{\varepsilon\} = L$
- $L/(L_1 \cup L_2) = L/L_1 \cup L/L_2$
- $L_1/L_2 = (L_2^R \backslash L_1^R)^R$

Úkol 33 *Zjistěte, zda platí* $L/(L_1 \cap L_2) = L/L_1 \cap L/L_2$.
Dále zjistěte, zda operace "/" je asociativní.

Tvrzení 5.0.13 L *je regulární právě když* L^R *je regulární.*

Důkaz. Idea: (u NKA) zaměníme počáteční stavy s přijímajícími a obrátíme šipky.
Formálně:
Nechť $L = L(A)$ pro NKA $A = (Q, \Sigma, \delta, I, F)$.
Definujme NKA $A' = (Q, \Sigma, \delta', F, I)$ tak, že pro vš. $q_1, q_2 \in Q$, $a \in \Sigma$: $q_2 \in \delta'(q_1, a) \Leftrightarrow q_1 \in \delta(q_2, a)$.
Pak lze snadno ukázat, že $L(A') = L^R$.
Konec Důkazu

Zkuste dále sestavit důkaz uzavřenosti třídy regulárních jazyků vůči kvocientům:

Tvrzení 5.0.14 *Jestliže* L_1, L_2 *jsou regulární, pak také* $L_2 \backslash L_1$ *a* L_1 / L_2 *jsou regulární.*

Návod:
Nechť $L_1 = L(A_1)$, kde $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, a $L_2 = L(A_2)$, kde $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$. Pro $q \in Q_1$ označme B_q automat $B_q = (Q_1, \Sigma, \delta_1, q, F_1)$ a

C_q automat $C_q = (Q_1, \Sigma, \delta_1, q_{01}, \{q\})$. Definujme dále $U = \{\, q \in Q_1 \mid \exists w \in \Sigma^* : w \in L(A_2) \wedge \delta_1(q_{01}, w) = q \,\}$; jinak řečeno: $q \in U \iff L(A_2) \cap L(C_q) \neq \emptyset$. (Zdůvodněte, proč existuje algoritmus rozhodující příslušnost k množině U.) Ukažte nyní, že $L_2 \backslash L_1 = \bigcup_{q \in U} L(B_q)$.

Úkol 34 *Uvažujme automaty A_1, A_2 zadané tabulkami:*

A_1

	a	b
$\rightarrow q_1$	q_2	q_3
$\leftarrow q_2$	q_2	q_4
$\leftarrow q_3$	q_5	q_3
q_4	q_2	q_4
q_5	q_5	q_3

A_2

	a	b
$\rightarrow r_1$	r_2	r_1
r_2	r_2	r_3
$\leftarrow r_3$	r_2	r_1

Zkonstruujte obecně použitelným algoritmem KA A rozpoznávající jazyk $L(A) = L(A_1)/L(A_2)$ (pravý kvocient). Poté se snažte jazyk $L(A)$ co nejjednodušeji charakterizovat (podmínkou, kterou splňují slova do něj patřící).

Kapitola 6

Regulární výrazy a jejich ekvivalence s konečnými automaty.

Již dříve jsme zmínili, že konečný automat "stojí v pozadí" mj. při vyhledávání vzorku v textu (např. při vyhledávání webovských stránek, které daný vzorek obsahují v záhlaví). Máme teď na mysli obecnější případ než je např. hledání konkrétního slova, a sice případ, kdy vzorek je specifikován (booleovskou) kombinací jednoduchých podmínek. Např. si lze představit, že výrazem "(česk∗ & sloven∗) ∨ (česk∗ & němec∗)" zadáváme (v nějakém systému) přání nalézt všechny dokumenty, které zároveň obsahují slovo začínající na "česk" a slovo začínající na "sloven" nebo zároveň obsahují slovo začínající na "česk" a slovo začínající na "němec".

Na výrazy podobné uvedenému lze pohlížet jako na popis (reprezentaci) určitého jazyka – reprezentovány jsou ty posloupnosti písmen (v "reálu" např. dokumenty, v našich pojmech jim říkáme prostě slova), které danému výrazu vyhovují; všimněte si, že takto reprezentovaný jazyk je pak obvykle nekonečný.

Teď si uvedeme definici tzv. regulárních výrazů a způsobu, jakým reprezentují jazyky (jak jste uhodli podle názvu, ukáže se, že regulárními výrazy lze reprezentovat právě regulární jazyky). Poznamenejme, že v různých softwarových systémech se setkáme s různými modifikacemi, nazvanými třeba také "regulární výrazy". V našem kursu (tak jako obecně v teorii jazyků a automatů) myslíme regulárními výrazy pouze pojem vymezený následující definicí.

Definice 6.1 *Množinu $RV(\Sigma)$ regulárních výrazů nad abecedou Σ definujeme jako nejmenší množinu slov v abecedě $\Sigma \cup \{\emptyset, \varepsilon, +, \cdot, ^*, (,)\}$ (předpokládáme $\emptyset, \varepsilon, +, \cdot, ^*, (,) \notin \Sigma$) splňující tyto podmínky:*

- $\emptyset \in RV(\Sigma)$, $\varepsilon \in RV(\Sigma)$, *pro každé* $a \in \Sigma$ *je* $a \in RV(\Sigma)$,
- *jestliže* $\alpha, \beta \in RV(\Sigma)$, *pak také* $(\alpha + \beta) \in RV(\Sigma), (\alpha \cdot \beta) \in RV(\Sigma), (\alpha^*) \in RV(\Sigma)$.

Definice 6.2 *Regulární výraz α reprezentuje jazyk – který označujeme $[\alpha]$ – tímto způsobem: $[\emptyset] = \emptyset$, $[\varepsilon] = \{\varepsilon\}$, $[a] = \{a\}$ a dále $[(\alpha + \beta)] = [\alpha] \cup [\beta]$, $[(\alpha \cdot \beta)] = [\alpha] \cdot [\beta]$, $[(\alpha^*)] = [\alpha]^*$.*

Poznámka. Při zápisu regulárních výrazů obvykle vynecháváme zbytečné závorky (asociativita operací, vnější pár závorek) a tečky pro zřetězení; další závorky lze vynechat díky dohodnuté prioritě operací: * váže silněji než $\cdot$, která váže silněji než $+$. Např. místo $((((0 \cdot 1)^* \cdot 1) \cdot (1 \cdot 1)) + ((0 \cdot 0) + 1)^*)$ obvykle napíšeme $(01)^*111 + (00 + 1)^*$.

Úkol 35 *Zadejte regulárním výrazem jazyk*
$L = \{ w \in \{0,1\}^* \mid$ *ve w je sudý počet nul a každá jednička je bezprostředně následována nulou* $\}$

Úkol 36 *Zjistěte, zda platí*
$[(011 + (10)^*1 + 0)^*] = [011(011 + (10)^*1 + 0)^*]$
$[((1 + 0)^*100(1 + 0)^*)^*] = [((1 + 0)100(1 + 0)^*100)^*]$

Úkol 37 *Procvičujte si regulární výrazy tím, že jimi popíšete některé regulární jazyky, s nimiž se v našem textu (včetně úkolů) setkáváte.*

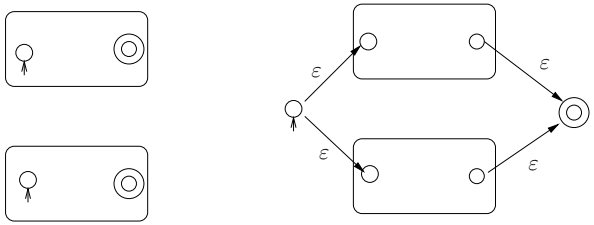
Přednáška 5

Již jsme avizovali, že vzhledem k popisu jazyků jsou regulární výrazy stejně silný prostředek jako konečné automaty. Jeden směr je zřejmý:

Pro jazyky $\emptyset$, $\{\varepsilon\}$, $\{a\}$ lze triviálně zkonstruovat rozpoznávající KA. Na základě příslušných konstrukcí v důkazech uzavřenosti třídy regulárních jazyků na ConceptUsageregulární operaceregulární operceRegOp pak snadno sestavíme algoritmus, který k libovolnému regulárnímu výrazu α sestrojí (zobecněný) nedeterministický konečný automat A rozpoznávající jazyk $[\alpha]$. Všimněte si, že počet stavů A bude přitom v zásadě odpovídat délce α.

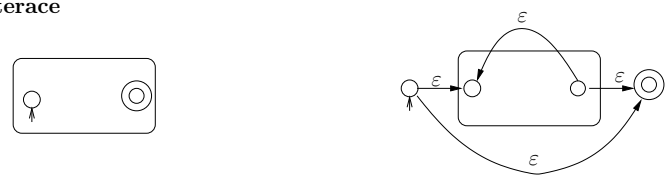
Úkol 38 *Myšlenky algoritmu převodu $RV \to ZNKA$ jsou načrtnuty na obrázku 6.1 – algoritmus k danému regulárnímu výrazu sestrojí ZNKA s jediným počátečním a jediným přijímajícím stavem.*
Aplikujte tento algoritmus na regulární výraz $((01^*0 + 101)^*100 + (11)^*0)^*01$.

Sjednocení



Zřetězení

Iterace



Obrázek 6.1: Konstrukce ZNKA k regulárnímu výrazu

Věta 6.0.15 *Regulárními výrazy lze reprezentovat právě regulární jazyky.*

Důkaz. Jelikož pro jazyky $\emptyset$, $\{\varepsilon\}$, $\{a\}$ lze triviálně zkonstruovat rozpoznávající KA, a třída regulárních jazyků je uzavřena na regulární operace (sjednocení, zřetězení, iterace), je zřejmý fakt, že ke každému regulárnímu výrazu α existuje (lze zkonstruovat) KA A tž. $L(A) = [\alpha]$ (tím jsme se zabývali v úkolu 38).

Technicky složitější je důkaz opačného směru, že totiž ke každému KA A existuje (opět lze zkonstruovat) regulární výraz α tž. $[\alpha] = L(A)$.

(Velmi) hrubá idea:

Slovo w je přijímáno automatem A právě když v grafu automatu existuje cesta (ohodnocená) w začínající v počátečním stavu q_0 a končící v "prvním" přijímajícím stavu nebo v "druhém" přijímajícím stavu atd. – v onom "nebo" lze snadno rozpoznat sjednocení jazyků.

Když cesta w vede z q_0 do (pevně zvoleného přijímajícího) q_A, tak buď je to "přímá cesta" – na níž se žádný stav neopakuje – nebo vznikne z přímé cesty "vložením cyklů". Přímých cest z q_0 do q_A je samozřejmě konečně mnoho (každá je nutně kratší než je počet stavů automatu), rozmístění cyklů je také konečně mnoho, a cykly lze iterovat. Stačí tedy "regulárně" popsat cykly a budeme hotovi. Elementárních cyklů (těch, které neobsahují kratší cyklus) je sice konečně mnoho, ale lze je různě kombinovat; to způsobuje, že ono regulární popsání vůbec není nabíledni a je velmi žádoucí podat přesvědčivý důkaz. Vhodný je např. induktivní důkaz, jenž je stručně načrtnut níže.

Induktivní důkaz:

Nechť $L = L(A)$ pro KA $A = (Q, \Sigma, \delta, q_1, F)$, kde $Q = \{q_1, q_2, \ldots, q_n\}$. Pro vš. $i, j \in \{1, 2, \ldots, n\}$ definujme $R_{ij} = \{w \in \Sigma^* \mid \delta(q_i, w) = q_j\}$ (tj. jako množinu slov, které převedou A ze stavu q_i do stavu q_j).

Dokážeme-li, že každá množina R_{ij} $(i, j \in \{1, 2, \ldots, n\})$ je reprezentovatelná regulárním výrazem, jsme hotovi – je totiž $L = \bigcup_{q_i \in F} R_{1i}$.

Zvolme nyní pevně i, j a uvažujme množinu R_{ij}. Pro $k \in \{0, 1, 2, \ldots, n\}$ definujme R^k_{ij} jako množinu slov, které převedou A ze stavu q_i do stavu q_j, přičemž všechny průběžné stavy mají index nejvýše rovný k. ($R^k_{ij} = \{w \in R_{ij} \mid \forall u, v : (u \neq \varepsilon \wedge v \neq \varepsilon \wedge w = uv \wedge \delta(q_i, u) = q_m) \Longrightarrow m \leq k :\}$.)

Ukážeme-li, že pro každé k je množina R^k_{ij} reprezentovatelná regulárním výrazem, budeme hotovi – je totiž $R_{ij} = R^n_{ij}$. To ovšem lze ukázat indukcí podle k. Základem indukce je triviální fakt

$R^0_{ij} \subseteq \Sigma \cup \{\varepsilon\}$. Indukční krok je zřejmý ze vztahu

$$R^{k+1}_{ij} = R^k_{ij} \ \cup \ (\ R^k_{i,k+1} \ (R^k_{k+1,k+1})^* \ R^k_{k+1,j} \ )$$

Konec Důkazu

Úkol 39 *Sestrojte regulární výraz reprezentující jazyk rozpoznávaný automatem zadaným uvedenou tabulkou.*

	a	b
$\rightarrow 1$	2	1
2	2	3
$\leftarrow 3$	2	1

Aplikujte přitom postup z předchozího důkazu, při němž se postupně konstruují výrazy reprezentující množiny $R^k_{i,j}$.

Dříve jsme rovnou zavedli pojem *regulární jazyk* jako synonymum pro *jazyk rozpoznatelný konečným automatem*. Pro úplnost dodejme, že v literatuře lze najít následující definici regulárních jazyků; ve světle výsledků uvedených výše je zřejmé, že obsah obou definic je totožný.

Třída $RJ(\Sigma)$ regulárních jazyků nad abecedou Σ je nejmenší třída jazyků nad abecedou Σ, která obsahuje tzv. *elementární jazyky* a je uzavřena na ConceptUsageregulární operaceregulární operceRegOp, tzn.:

- elementární jazyky, tj. $\emptyset$ a $\{a\}$ (pro každé $a \in \Sigma$), patří do $RJ(\Sigma)$,
- jestliže $L_1, L_2 \in RJ(\Sigma)$, pak také $L_1 \cup L_2 \in RJ(\Sigma)$,
- jestliže $L_1, L_2 \in RJ(\Sigma)$, pak také $L_1 \cdot L_2 \in RJ(\Sigma)$,
- jestliže $L \in RJ(\Sigma)$, pak také $L^* \in RJ(\Sigma)$.

*Poznámka.*Jak plyne z uzávěrových vlastností třídy regulárních jazyků, mohli bychom regulární výrazy obohatit např. symboly pro průnik a doplněk (třeba &, ¬, přičemž $[(\alpha\&\beta)] = [\alpha] \cap [\beta]$, $[\neg(\alpha)] = \Sigma^* - [\alpha]$ – abeceda Σ musí být zřejmá z kontextu), aniž se zvětší jejich vyjadřovací síla. Zápis jazyka se tak někdy zkrátí (např. místo $(0 + 1)^*1(0 + 1)^*$ lze psát $\neg(0^*)$; ztrácí se ale např. vlastnost přímočarého převodu RV → ZNKA zmíněného výše. To je jeden z důvodů proč průnik ani doplněk neřadíme k (standardním) ConceptUsageregulárním operacímregulární operceRegOp.

*Poznámka.*Důkazy matematickou indukcí (např. u věty 3.0.7 a dalších) jsme dosud v textu podrobně neprováděli. Příslušná tvrzení byla očividná a důkaz indukcí je u nich v podstatě jen rutinní cvičení (ovšem velmi užitečné !). Induktivní důkaz jsme skutečně provedli až nyní u věty 6.0.15, protože zde se bez něj opravdu těžko lze obejít (jak byste měli potvrdit, pokud jste se poctivě snažili větu "nahlédnout" a dokonale se přesvědčit o její platnosti).

Regulární výrazy nám poskytují další možnost reprezentace regulárních jazyků. Toho lze mj. také využít k elegantním důkazům některých dalších uzávěrových vlastností třídy regulárních jazyků; zde to ilustrujeme na příkladu tzv. regulární substituce.

Definice 6.3 *Nechť Σ je abeceda a pro každé $a \in \Sigma$ je dán jazyk $\sigma(a)$ v abecedě Δ_a. Položme $\sigma(\varepsilon) = \{\varepsilon\}$ a $\sigma(uv) = \sigma(u)\sigma(v)$ pro vš. $u, v \in \Sigma^*$. Potom zobrazení $\sigma : \Sigma^* \to \mathcal{P}(\Delta^*)$, kde $\Delta = \bigcup_{a\in\Sigma} \Delta_a$, se nazývá* substituce.

Pro každý jazyk $L \subseteq \Sigma^*$ pak definujeme $\sigma(L) = \bigcup_{w\in L} \sigma(w)$ a říkáme, že jazyk $\sigma(L)$ vznikl z jazyka L substitucí σ.

Substituce σ, u níž pro každé $a \in \Sigma$ obsahuje jazyk $\sigma(a)$ právě jedno slovo, se nazývá homomorfismus. *Homomorfismus h lze pak tedy považovat za zobrazení typu $h : \Sigma^* \to \Delta^*$ (které splňuje $h(\varepsilon) = \varepsilon, h(uv) = h(u)h(v)$).*

Tvrzení 6.0.16 *Nechť Σ je abeceda a σ* regulární substituce, *tzn. $\sigma(a)$ je regulární jazyk pro každé $a \in \Sigma$. Potom pro lib. regulární jazyk $L \subseteq \Sigma^*$ platí, že $\sigma(L)$ je rovněž regulárním jazykem.*

Důkaz. Nechť reg. výraz α reprezentuje L a nechť α_a reprezentuje $\sigma(a)$, pro každé $a \in \Sigma$. Dá se snadno ověřit, že dosadíme-li do α za každý výskyt symbolu a reg. výraz α_a, dostaneme regulární výraz reprezentující $\sigma(L)$.
Konec Důkazu

Úkol 40 *Uvedené tři tabulky nedeterministických automatů zadávají po řadě regulární jazyky L_1, L_2, L_3.*

	a	*b*
$\to A$	*A,B,C*	-
$\to B$	-	*A,B,C*
$\leftarrow C$	*B*	*C*

	0	*1*
$\leftrightarrow D$	*E*	*F*
$\leftarrow E$	*E,F*	*D*
F	*D*	*F*

	0	*1*
$\to G$	*G*	*H*
$\leftarrow H$	*H*	*G*

Definujte regulární substituci σ předpisem $\sigma(a) = L_2$, $\sigma(b) = L_3$. Sestrojte regulární výraz popisující jazyk $\sigma(L_1)$.

Úkol 41 *K hlubšímu pochopení některých pojmů můžete zkusit zjistit, zda platí následující obecné vztahy; h označuje homomorfismus, $\backslash$ levý kvocient.*

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

- $\{w\}\backslash(L_1 \cup L_2) = \{w\}\backslash L_1 \cup \{w\}\backslash L_2$
- $\{w\}\backslash(\Sigma^* - L) = \Sigma^* - \{w\}\backslash L$
- $L_2(L_2\backslash L_1) = L_1$

Kapitola 7

Minimální konečné automaty a algoritmus minimalizace.

Přednáška 6

Vezmeme-li v úvahu praktické (implementační) hledisko, jistě není třeba sáhodlouze motivovat otázku možné minimalizace daného konečného automatu. Ukážeme, že odpověď je v našem případě velmi potěšující: existuje algoritmus (dokonce "rychlý" algoritmus), který k zadanému KA sestrojí ekvivalentní automat s nejmenším možným počtem stavů.

Definice 7.1 *Dva konečné* automaty A_1, A_2 *nazveme* jazykově ekvivalentní, *stručněji* ekvivalentní, *jestliže rozpoznávají týž jazyk, tj. jestliže* $L(A_1) = L(A_2)$.

Konečný automat nazveme minimálním automatem jestliže neexistuje automat, který by s ním byl ekvivalentní a měl by menší počet stavů.

Naším cílem nyní bude dokázat tuto větu:

Věta 7.0.17 *Existuje algoritmus, který k zadanému konečnému automatu A sestrojí minimální automat ekvivalentní s A.*

Úkol 42 *Velmi užitečné by bylo, kdybyste nejdříve sami zkusili (nějak) najít minimální automat ekvivalentní zde zadanému.*
V každém případě je vhodné si třeba na tomto příkladu ilustrovat dále zaváděné pojmy; konkrétně začněte konstrukcí rozkladů množiny stavů podle ekvivalencí $\overset{0}{\sim}, \overset{1}{\sim}, \overset{2}{\sim}, \ldots$, *které jsou definovány až trochu dálee v textu.*

	a	b
$\rightarrow 1$	2	3
2	2	4
$\leftarrow 3$	3	5
4	2	7
$\leftarrow 5$	6	3
$\leftarrow 6$	6	6
7	7	4
8	2	3
9	9	4

Ukážeme nejdříve hlavní ideu algoritmu. Uvažujme konečný automat $A = (Q, \Sigma, \delta, q_0, F)$, rovnou bez nedosažitelných stavů – ty bychom jinak prostě odstranili, aniž změníme rozpoznávaný jazyk (vzpomeňte si, že touto otázkou jsme se již zabývali dříve). Pro každý stav $q \in Q$ definujme

$$L(q) = L(A_q), \text{ kde } A_q = (Q, \Sigma, \delta, q, F)$$

($L(q)$ je tedy jazyk rozpoznávaný automatem, jenž vznikne z A prohlášením stavu q za počáteční, tedy množina těch slov, které převedou automat A ze stavu q do některého stavu v F).

Nechť nyní pro dva různé stavy q_1, q_2 platí $L(q_1) = L(q_2)$. Teď přijde ta idea: jeden z těchto stavů, např. q_2, můžeme z automatu A vypustit, přičemž šipky do něj směřující (v terminologii grafu automatu) přesměrujeme do q_1. Musíme dodat, že pokud q_2 byl počátečním (v automatu A), prohlásíme za nový počáteční stav q_1. Snadno se lze přesvědčit, že takto vzniklý automat je ekvivalentní s (původním) automatem A, přičemž má méně stavů – a nemá nedosažitelné stavy, když A je neměl. (Přesvědčte se !)

Vedle odstranění nedosažitelných stavů tak máme k dispozici další metodu, jak zmenšovat KA při zachování rozpoznávaného jazyka. Opakované použití této metody nás očividně přivede k automatu, který je ekvivalentní s původním a je to tzv. redukovaný automat, což znamená, že nemá nedosažitelné stavy a pro každé jeho dva různé stavy q_1, q_2 platí $L(q_1) \neq L(q_2)$.

Tento výsledný automat lze elegantně popsat jako tzv. podílový automat podle ekvivalence $\sim$, kde relace $\sim$ je zavedena na množině stavů KA takto:

$$q \sim q' \Longleftrightarrow_{df} L(q) = L(q')$$

(je to samozřejmě jiná relace než ekvivalence mezi konečnými automaty zavedená výše).

*Vsuvka.*Připomeňme, že relace ρ na množině M je *ekvivalence* právě tehdy, když je reflexivní ($\forall x \in M : x\rho x$), symetrická ($\forall x, y \in M : x\rho y \Rightarrow y\rho x$) a tranzitivní ($\forall x, y, z \in M : (x\rho y \wedge y\rho z) \Longrightarrow x\rho z$). Ekvivalence ρ definuje na M *rozklad* $\{\, [x]_\rho \mid x \in M \,\}$, tj. systém vzájemně disjunktních množin (neboli tříd ekvivalence), jejichž sjednocením je M, přičemž s každým $x \in M$ jsou v příslušné třídě obsaženy právě prvky s ním ekvivalentní ($[x]_\rho = \{y \mid x\rho y\}$).

Formální popis konstrukce podílového automatu:

Lemma 7.0.18 *K libovolnému konečnému automatu existuje ekvivalentní redukovaný automat.*

Důkaz. Mějme automat $A = (Q, \Sigma, \delta, q_0, F)$ bez nedosažitelných stavů. Značením $[q]$ ($q \in Q$) označujeme třídy ekvivalence $\sim$ obsahující q (tj. $[q] = \{p \mid p \sim q\}$).

Nyní k A definujme tzv. podílový automat podle ekvivalence $\sim$, označený $A_\sim$, takto: $A_\sim = (Q_\sim, \Sigma, \delta_\sim, [q_0], F_\sim)$, kde $Q_\sim = \{\, [q] \mid q \in Q \,\}$, $F_\sim = \{\, [q] \mid q \in F \,\}$ a $\delta_\sim([q], a) = [\delta(q, a)]$ (pro vš. $q \in Q$, $a \in \Sigma$).

Korektnost definice plyne z faktu $p \sim q \Rightarrow (p \in F \Leftrightarrow q \in F)$ a z faktu $p \sim q \Rightarrow \delta(p, a) \sim \delta(q, a)$.

Snadno ověříme, že $\delta(q, w) = q' \Leftrightarrow \delta_\sim([q], w) = [q']$ (např. provedením důkazu indukcí podle délky w), z čehož ihned vyvodíme ekvivalenci automatů A a $A_\sim$ (tj. $L(A) = L(A_\sim)$), i to, že $A_\sim$ nemá nedosažitelné stavy.
Konec Důkazu

K algoritmickému použití naší metody ovšem potřebujeme ukázat, že umíme pro libovolné stavy q_1, q_2 rozhodovat otázku, zda $L(q_1) = L(q_2)$. Pro tyto účely je vhodné definovat $L^{\leq i}(q)$ jako množinu všech slov z $L(q)$, které mají délku nejvýše i, a pak zavést na stavové množině automatu $A = (Q, \Sigma, \delta, q_0, F)$ relace $\overset{0}{\sim}, \overset{1}{\sim}, \overset{2}{\sim}, \ldots$ takto:

$$q_1 \overset{i}{\sim} q_2 \Longleftrightarrow_{df} L^{\leq i}(q_1) = L^{\leq i}(q_2)$$

Jinak řečeno: pro stavy q_1, q_2 platí $q_1 \overset{i}{\sim} q_2$ právě když je nelze rozlišit žádným slovem délky nejvýše i (tj. pro každé slovo $w \in \Sigma^*$, $|w| \leq i$, je buď $\delta(q_1, w) \in F$, $\delta(q_2, w) \in F$ nebo $\delta(q_1, w) \notin F$, $\delta(q_2, w) \notin F$). Je očividné, že relace $\overset{i}{\sim}$ jsou ekvivalence a platí $\overset{0}{\sim} \supseteq \overset{1}{\sim} \supseteq \overset{2}{\sim} \supseteq \ldots$ (tedy $q_1 \overset{i+1}{\sim} q_2 \Longrightarrow q_1 \overset{i}{\sim} q_2$, neboli $\overset{i+1}{\sim}$ je zjemněním relace $\overset{i}{\sim}$).

Samozřejmě vidíme, že $q_1 \overset{0}{\sim} q_2$ právě když buď $q_1 \in F$, $q_2 \in F$ nebo $q_1 \notin F$, $q_2 \notin F$.

Dále je zřejmé, že dva stavy jsou rozlišitelné slovem délky nejvýše $i+1$ právě když jsou rozlišitelné slovem délky nejvýše i nebo existuje $a \in \Sigma$, které je převede do dvojice stavů rozlišitelných slovem délky nejvýše i; v obměněné podobě to můžeme formálně vyjádřit takto:

$$q_1 \overset{i+1}{\sim} q_2 \Longleftrightarrow q_1 \overset{i}{\sim} q_2 \wedge (\forall a \in \Sigma :\ \delta(q_1, a) \overset{i}{\sim} \delta(q_2, a)) \tag{7.1}$$

Všimněme si teď, že ekvivalence $\overset{0}{\sim}$ rozloží množinu stavů Q na dvě třídy F, $Q - F$ (když jsou obě neprázdné). Také víme, že $\overset{i+1}{\sim}$ rozloží Q na stejně nebo více tříd než $\overset{i}{\sim}$, a ze vztahu (7.1) je zřejmé, že pokud $\overset{i}{\sim} = \overset{i+1}{\sim}$, pak $\overset{i}{\sim} = \overset{i+1}{\sim} = \overset{i+2}{\sim} = \ldots = \sim$.

Jelikož při počtu stavů n ($|Q| = n$) nemůže existovat rozklad na více než n tříd, je dokonce zřejmé, že rovnost $\overset{i}{\sim} = \overset{i+1}{\sim}$ musí určitě nastat pro nějaké $i \leq n - 2$. Stačí tedy postupně konstruovat relace, resp. rozklady podle relací, $\overset{0}{\sim}, \overset{1}{\sim}, \overset{2}{\sim}, \ldots$, až zjistíme $\overset{i}{\sim} = \overset{i+1}{\sim}$ – víme, že pak $\overset{i}{\sim} = \sim$, a můžeme tak pro libovolné stavy q_1, q_2 rozhodovat, zda $L(q_1) = L(q_2)$ (zjištěním, zda jsou ve stejné třídě zkonstruovaného rozkladu podle $\sim$).

Máme tedy algoritmický postup, jak k danému automatu A zkonstruovat ekvivalentní automat B, který je redukovaný (lemma 7.0.18 tedy platí konstruktivně) – přitom když A je redukovaný, pak B je totožný s A, v opačném případě má B méně stavů než A. Je ale B ten nejmenší možný automat mezi automaty ekvivalentními s A ? Kladnou odpověď jednoduše vyvodíme z následujícího jednoduchého lemmatu (na něj se de facto odkazujeme i v důkazu 7.0.18).

Lemma 7.0.19 *Mějme dva KA $A = (Q, \Sigma, \delta, q_0, F)$ a $A' = (Q', \Sigma, \delta', q'_0, F')$ Jestliže pro stav q automatu A a stav q' automatu A' platí $L(q) = L(q')$, pak pro každé $a \in \Sigma$ je $L(\delta(q, a)) = L(\delta'(q', a))$.*

Důkaz. Ukážeme, že když $L(\delta(q, a)) \neq L(\delta'(q', a))$, pak $L(q) \neq L(q')$. Když např. $u \in L(\delta(q, a))$ a $u \notin L(\delta'(q', a))$, pak nutně $au \in L(q)$ a $au \notin L(q')$.
Konec Důkazu

Uvědomte si, že z toho snadno plyne, že dva *redukované* automaty $A = (Q, \Sigma, \delta, q_0, F)$ a $A' = (Q', \Sigma, \delta', q'_0, F')$, které jsou ekvivalentní (tj. $L(q_0) = L(q'_0)$), mají jednak stejný počet stavů a dokonce jsou vlastně totožné (lépe řečeno: jsou izomorfní, tj. jeden dostaneme z druhého pouhým přejmenováním stavů). Teď už je zřejmé, že

minimální automat znamená totéž co redukovaný automat.

Dokázali jsme tak vlastně nejen větu 7.0.17, ale i

Věta 7.0.20 *Minimální automat ekvivalentní s daným KA A je určen jednoznačně.*

Víme, že tím "jednoznačně" míníme "jednoznačně, až na pojmenování stavů". Libovůli v onom pojmenování ovšem lze odstranit požadavkem převodu do normovaného tvaru, který jsme již definovali dříve.

Úkol 43 *Zjistěte všechny dvojice stavů q, q' u následujících dvou automatů (tedy $q, q' \in \{0, 1, 2, \ldots, 9\}$), pro něž $L(q) = L(q')$.*

	a	b
$\rightarrow 0$	0	1
$\leftarrow 1$	1	2
$\leftarrow 2$	3	1
3	2	4
4	2	3

	a	b
$\rightarrow 5$	5	6
6	7	5
$\leftarrow 7$	7	9
8	9	8
$\leftarrow 9$	8	7

Úkol 44 *Sestrojte minimální (deterministický) konečný automat, který rozpoznává tentýž jazyk jako NKA zadaný následující tabulkou (a převeďte ho do normovaného tvaru):*

	a	b
$\rightarrow q_0$	q_1, q_3	q_5
q_1	-	q_2
q_2	q_F	-
q_3	-	q_4
q_4	-	q_F
q_5	-	q_6
q_6	-	q_N
$\leftarrow q_F$	q_F	q_F
q_N	-	-

Úkol 45 *Sestrojte minimální (deterministické) konečné automaty, rozpoznávající jazyky reprezentované následujícími regulárními výrazy:*

- $(ab^*b + ab^*ab^*b + ab^*ab^*a)^*$
- $(a + bb)^* + (\,(b + c)^* \cdot (d + e)^*\,)^+$ *(kde pro jazyk L definujeme $L^+ = L + L^2 + L^3 + \ldots$ a pro reg. výraz α definujeme $[\alpha^+] = [\alpha]^+$)*

Všimněme si, že nyní (nejméně) dvěma různými způsoby umíme dokázat tuto důležitou větu:

Věta 7.0.21 *Existuje algoritmus, který pro lib. zadané KA A_1, A_2 rozhodne, zda $L(A_1) = L(A_2)$.*

Důkaz. Stačí k oběma KA sestrojit redukované automaty v normovaném tvaru a ty porovnat. Jiný důkaz plyne z partie o (konstruktivních) uzávěrových vlastnostech třídy regulárních jazyků, uvědomíme-li si, že $L(A_1) = L(A_2) \Longleftrightarrow (L(A_1) - L(A_2)) \cup (L(A_2) - L(A_1)) = \emptyset$ a připomeneme-li si větu 2.0.2.
Konec Důkazu

*Poznámka.*Všimněte si, že jsme dokázali, že pokud dva stavy automatu, který má n stavů ($n \geq 2$), nelze rozlišit slovem délky nejvýše $n - 2$, pak je nelze rozlišit vůbec (jestliže $L^{\leq n-2}(q) = L^{\leq n-2}(q')$, pak $L(q) = L(q')$). Tento fakt ukazuje, že pro rozhodování otázky, zda $L(q) = L(q')$, stačí probrat všechna slova do délky $n - 2$ (jichž je samozřejmě konečně mnoho). Ovšem algoritmus založený na této myšlence by byl pro praktické použití velmi nevhodný. (Proč ?)

Kapitola 8

Neregulární jazyky. Pumping lemma pro regulární jazyky.

Přednáška 7

Čtenáři by mělo být jasné, že ne každý jazyk je regulární (už jsme to dokonce dokázali při úvahách o mohutnostech množin: konečných automatů je spočetně mnoho, zatímco jazyků – už nad jednoprvkovou abecedou – je nespočetně mnoho).

Jak ale vypadá konkrétní neregulární jazyk ? Musí mít nějakou vlastnost, která neumožňuje rozpoznání libovolného jeho slova (tedy také libovolně dlouhého slova), máme-li pouze omezenou paměť a můžeme slovo pouze číst zleva doprava.

Uvažujme např. jazyk

$$L = \{a^j b^j \mid j \geq 0\}$$

(každé slovo tedy začíná úsekem a-ček, za nímž následuje stejně dlouhý úsek b-ček). Intuitivně je vidět, že při čtení slova zleva doprava nám "nezbývá nic jiného" než a-čka počítat a pak porovnat s počtem b-ček; k tomu nám ovšem předem omezená paměť nestačí, protože úsek a-ček může být libovolně dlouhý !

Ale pozor ! Tyto naše úvahy se nedají považovat za důkaz toho, že L je neregulární. Naše intuice nás může klamat, a třeba je to jen naší omezeností, že nevidíme způsob, jak se bez počítání a-ček můžeme obejít. Když by nám např. někdo tvrdil, že jazyk $\{ a^m \mid m$ je dělitelné třemi $\}$ není regulární, protože nezbývá nic jiného než a-čka spočítat a výsledek dělit třemi, vyvrátili

bychom mu to prostě předvedením konečného automatu, který tento jazyk rozpoznává – a tudíž se zde bez počítání a-ček lze obejít. Jak ale dokázat, že něco nelze ? Obvykle je klíčem vyvození logického sporu z předpokladu, že to lze.

U L bychom mohli postupovat takto: Předpokládejme, že L je rozpoznáván konečným automatem A; ten má nějaký (konečný) počet stavů, označme tento počet n. Automat A musí samozřejmě přijmout i slovo $a^n b^n$. Při čtení úseku a^n prochází postupně určitými stavy $q_0, q_1, q_2, \ldots, q_n$. Jelikož A má pouze n stavů, nutně se nějaký stav zopakuje, tedy $q_i = q_j$ pro nějaké i, j, kde $0 \leq i < j \leq n$. Pak ovšem slovo vzniklé zopakováním úseku mezi q_i a q_j, tedy slovo $a^i a^{j-i} a^{j-i} a^{n-j} b^n$, je automatem A přijímáno (proč ?); to ovšem nepatří do L a přivedli jsme tak ke sporu předpoklad, že A rozpoznává L. Všimněte si také, že A by musel rozpoznávat nejen slovo vzniklé jedním zopakováním příslušného úseku, ale také slova vzniklá libovolným "napumpováním" tohoto úseku, tedy slova tvaru $a^i a^{j-i} a^{j-i} \ldots a^{j-i} a^{n-j} b^n$; speciálním případem je pak vypuštění úseku, u nás slovo $a^i a^{n-j} b^n$.

Uvedené úvahy se snadno dají zobecnit. Velmi zhruba řečeno:

> v každém "dostatečně dlouhém" slově regulárního jazyka L existuje "krátké" neprázdné podslovo "blízko začátku", jehož vynecháním či "pumpováním" dostáváme vždy slova jazyka L

Formálně (a přesně) to vyjadřuje následující věta, již si čtenář teď už jistě sám snadno dokáže ("nápověda": za n, jehož existenci věta deklaruje, si pro konkrétnost můžete dosadit např. počet stavů minimálního automatu rozpoznávajícího L).

Věta 8.0.22 (Pumping lemma.) *Nechť L je regulární jazyk. Pak nutně existuje n tž. každé slovo $z \in L$, $|z| \geq n$, lze psát $z = uvw$, kde $|uv| \leq n$, $|v| \geq 1$ a pro vš. $i \geq 0$ je $uv^i w \in L$.*

Všimněte si, jak se střídají kvantifikátory: Je-li L regulární, pak (neboli ($\forall L$ tž. L je regulární) :)

$(\exists n)\ (\forall z$ tž. $z \in L, |z| \geq n)\ (\exists\, u, v, w$ tž. $z = uvw, |uv| \leq n, |v| \geq 1)\ (\forall i \geq 0) : uv^i w \in L$

Poznámka. Více formálně bychom místo $(\forall x$ tž. $A)\, B$ psali $(\forall x : A \Rightarrow B)$ a místo $(\exists x$ tž $A)\, B$ bychom psali $(\exists x : A \wedge B)$.

Je užitečné představit si hru dvou hráčů A a B, kteří mají zadán (nějaký) jazyk L a hrají takto:

1. A zvolí $n \in \mathcal{N}$

2. B zvolí slovo z tž. $z \in L$ a $|z| \geq n$ (neexistuje-li takové slovo, A vyhrál)

3. A zvolí u, v, w tž. $z = uvw$, $|uv| \leq n$ a $|v| \geq 1$

4. B zvolí $i \geq 0$

5. *Výsledek:* je-li $uv^i w \in L$, pak vyhrál A, v případě $uv^i w \notin L$ vyhrál B.

Je zřejmé, že je-li L regulární, pak A má vítěznou strategii v uvedené hře. Jinak řečeno:

Má-li B vítěznou strategii, pak L není regulární.

A právě navržení vítězné strategie hráče B je častým prostředkem k důkazu toho, že uvažovaný jazyk je neregulární.
Pro výše zkoumaný $L = \{a^j b^j \mid j \geq 0\}$ můžeme vítěznou strategii hráče B formulovat takto.

1. A zvolí (libovolné) $n \in \mathcal{N}$

2. B zvolí $z = a^n b^n$

3. A zvolí libovolné u, v, w tž. $z = uvw$, $|uv| \leq n$ a $|v| \geq 1$, tedy $u = a^j$, $v = a^k$ pro nějaké j, k tž. $j + k \leq n$, $k \geq 1$

4. B: zvolí $i = 0$ (lze také kterékoli $i \geq 2$)

5. Jelikož $a^j a^{n-(j+k)} b^n \notin L$, B vyhrává.

Všimněme si, že uvedená vítězná strategie hráče B pro jazyk $L_1 = \{a^j b^j \mid j \geq 0\}$ je rovněž vítěznou strategií pro jazyk $L_2 = \{\, w \in \{a, b\}^* \mid |w|_a = |w|_b \,\}$ (připomeňme, že $|w|_a$ označuje počet výskytů symbolu a ve slově w); konečný automat tedy nemůže rozpoznávat slova, v nichž jsou počty a-ček a b-ček stejné.
Z faktu, že L_1 není regulární, lze ovšem neregularitu jazyka L_2 elegantně prokázat také takto:
Kdyby L_2 byl regulární, byl by regulární i jazyk $L_2 \cap a^*b^*$, jelikož třída regulárních jazyků je uzavřena na průnik (a a^*b^* je očividně regulární). Ovšem je zřejmé, že $L_2 \cap a^*b^* = L_1$, a L_1 regulární není. Předpoklad, že L_2 je regulární je takto přiveden ke sporu, což znamená, že L_2 regulární není.
Správně bychom měli psát $[a^*b^*]$ místo a^*b^*, jelikož se odkazujeme k jazyku reprezentovanému daným regulárním výrazem. Protože ale nemůže dojít k nedorozumění, značení si takto zjednodušujeme.

Samozřejme musíme ale být opatrní při posuzování (ne)regularity jazyka na základě jeho specifikace. Např. podobnost specifikace $L_3 = \{\, w \in \{a, b\}^* \mid$ počty podslov ab a ba ve w jsou stejné $\}$ s výše uvedenou specifikací jazyka L_2 může nabuzovat dojem, že L_3 je rovněž neregulární.

Úkol 46 *Zjistěte, zda jazyk L_3 je či není regulární. (Své zjištění dokažte.)*

Doplnění Čtenáře možná napadla otázka, zda pumping lemma přesně charakterizuje regulární jazyky, tj. zda pro jakýkoli neregulární jazyk má B vítěznou strategii. Není tomu tak, jak dokládá např.jazyk $L = \{a, b\}^* \cup \{c\}^+ \{a^j b^j \mid j \geq 0\}$, který splňuje podmínku v pumping lemmatu (A má pro něj vítěznou strategii) a přitom není regulární. (Pro připomenutí: $\{c\}^+ = \{\, c, cc, ccc, \ldots \,\}$.)
To, že uvedený L není regulární, jsme samozřejmě schopni dokázat, použijeme-li i jiné prostředky. Z předpokladu, že L je regulární, můžeme např. využitím uzavřenosti třídy regulárních jazyků vůči kvocientům a průniku vyvodit, že i

$$(\,\{c\}^+ \backslash L\,) \cap \{a, b\}^* = \{\, a^j b^j \mid j \geq 0 \,\}$$

je regulární – o tom jsme už ovšem ukázali, že regulární není, a dospíváme takto ke sporu.

Úkol 47 *Dokažte, že následující jazyky nejsou regulární (využijte pumping lemma a rozmyslete si formulace důkazů ve formě hry dvou hráčů)*

- $L_1 = \{\, 0^m 1^n 0^m \mid m, n \geq 0 \,\}$
- $L_2 = \{\, ww \mid w \in \{0, 1\}^* \,\}$
- $L_3 = \{\, w(w)^R \mid w \in \{0, 1\}^* \,\}$
- L_4 *je množina všech zápisů programů v Pascalu.*

Můžete zkusit též pro:

- $L_5 = \{\, 0^p \mid p$ *je prvočíslo* $\,\}$
- $L_6 = \{\, 0^n \mid n = k^2$ *pro nějaké* $k \geq 0 \,\}$

Úkol 48 *Uvažujte pumping lemma v tomto znění:*

> *Nechť L je regulární jazyk. Pak nutně existuje n tž. v každém slově $z \in L$ lze každé jeho podslovo x délky n ($|x| = n$) psát $x = uvw$, kde $|v| \geq 1$, přičemž pro $z = y_1 x y_2 = y_1 uvw y_2$ platí, že $y_1 uv^i w y_2 \in L$ pro vš. $i \geq 0$.*

Dokažte, že v tomto znění tvrzení také platí a vysvětlete, zda je obecnějším anebo speciálním případem dříve uvedené verze.

Kapitola 9

Další poznámky ke konečným automatům.

9.1 Dvoucestné konečné automaty

Ukazovali jsme, že nemůžeme rozpoznávat právě slova tvaru $a^j b^j$, máme-li omezenou paměť a čteme-li slova zleva doprava. Ve skutečnosti by nám nepomohla ani možnost vracet se k již přečtenému úseku slova – pokud symboly nemůžeme nijak měnit, tj. nemůžeme si ve slově nic poznačit. Dá se ukázat, že odpovídající model, tzv. dvoucestné konečné automaty, charakterizuje opět jen třídu regulárních jazyků, a to i v nedeterministické verzi; důkaz je ovšem technicky složitější.

9.2 Hledání vzorku v textu

Hledání všech výskytů vzorku v textu—tj. řetězce p (pattern) v (obvykle podstatně delším) řetězci t (text)—je velmi častá úloha každodenně probíhající v mnoha aplikacích. Proto je velmi podstatné, jakou časovou náročnost má použitý algoritmus. Naivní přístup, který nás asi napadne nejdříve (posouvající se "okénko" délky $|p|$ a kontrola, zda v okénku je p) vede k době úměrné $|p| \cdot |t|$. Podstatně lepší je tzv. "Knuth-Morris-Pratt algorithm", jehož doba běhu je úměrná $|p| + |t|$. Klíčovou myšlenkou tohoto algoritmu je tato:

Představme si přímočarou konstrukci nedeterministického konečného automatu, přijímajícího právě ta slova (v zadané abecedě), která mají sufix p. (Takový NKA by měl $|p| + 1$ stavů.) K tomuto NKA standardně zkonstru-

ovaný DKA (konstruujeme jen dosažitelné stavy) má také jen $|p|+1$ stavů. (Proč ?)

KMP-algoritmus pak obsahuje ještě další důležitou myšlenku (k použití onoho DKA není třeba konstruovat přechodovou funkci $\delta : Q \times \Sigma \to Q$, ale stačí použít jen jistou "failure" funkci $f : Q \to Q$), ale to v tomto textu dále nerozebíráme.

Úkol 49 *Sestrojte (deterministický) konečný automat, který je vhodný k použití pro vyhledávání řetězce ababaca ve slovech (textech) v abecedě* $\{a, b, c\}$.

9.3 Konečně stavový překladač

Již dříve jsme zmínili, že konečný automat jako rozpoznávač jazyka je speciálním případem konečně stavového zařízení, realizujícího jistou vstupně-výstupní funkci.

Jedním z těchto obecnějších modelů je tzv. zobecněný sekvenční stroj (generalized sequential machine):

Zobecněný sekvenční stroj M je dán parametry $M = (Q, \Sigma, \Delta, q_0, \delta, \rho)$, kde Q je konečná neprázdná množina *stavů*, Σ je konečná neprázdná množina zvaná vstupní abeceda, Δ je konečná neprázdná množina zvaná výstupní abeceda, $q_0 \in Q$ je *počáteční (iniciální) stav*, $\delta : Q \times \Sigma \to Q$ je *přechodová funkce* a $\rho : Q \times \Sigma \to \Delta^*$ je *výstupní* funkce.

Takový stroj M jistým způsobem definuje zobrazení ("překlad") $f_M : \Sigma^* \to \Delta^*$. Zkuste odhadnout jakým. (K modelu konečného automatu si přidejte výstupní pásku s tím, že v každém kroku daném přechodovou funkcí δ se na výstup připíše řetězec daný funkcí ρ.) Pro jazyk L v abecedě Σ lze pak přirozeně definovat jeho obraz $f_M(L)$ (v abecedě Δ); podobně pro jazyk L v abecedě Δ lze definovat jeho vzor (inverzní obraz) $f_M^{-1}(L)$ (v abecedě Σ). Dá se pak např. ukázat, že f_M i f_M^{-1} zachovávají regulární jazyky (tj. když L je regulární, tak $f_M(L)$ i $f_M^{-1}(L)$ jsou regulární.

Další zobecnění dostaneme, povolíme-li (mj.) nedeterminismus. Příslušné zařízení se pak nazývá konečně-stavový překladač (převaděč; v angličtině "finite-state transducer"), který pak nedefinuje funkci, ale obecněji relaci (podmnožinu $\Sigma^* \times \Delta^*$).

Kapitola 10

Bezkontextové gramatiky a jazyky

Přednáška 8

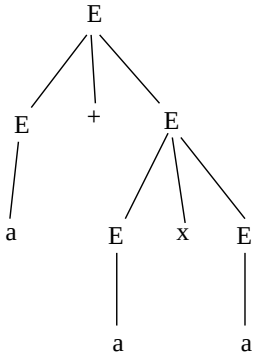
Uvažujme množinu všech aritmetických výrazů vytvořených z prvků abecedy $a, +, \times, (,)$ (číselné konstanty či proměnné teď nejsou podstatné, proto všechny reprezentujeme symbolem a). Příkladem takového výrazu je $a+a\times a$ nebo $(a+a)\times a$.

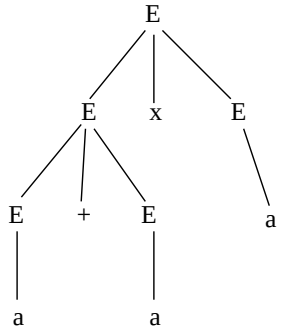
Čtenář jistě snadno dokáže, že se nejedná o regulární jazyk, nemůžeme jej tedy zadat regulárním výrazem či konečným automatem. V rámci v praxi užívaného popisu programovacích jazyků může být množina uvedených aritmetických výrazů zadaná těmito (přepisovacími) pravidly:

$$\langle EXPR\rangle \longrightarrow \langle EXPR\rangle + \langle EXPR\rangle$$
$$\langle EXPR\rangle \longrightarrow \langle EXPR\rangle \times \langle EXPR\rangle$$
$$\langle EXPR\rangle \longrightarrow (\langle EXPR\rangle)$$
$$\langle EXPR\rangle \longrightarrow a$$

Píšeme-li místo $\langle EXPR\rangle$ jen E a pravé strany pravidel se stejnou levou stranou soustředíme vedle sebe, přičemž je vzájemně oddělíme symbolem "|", vznikne:

$$E \longrightarrow E + E \mid E \times E \mid (E) \mid a \tag{10.1}$$





Jak vidno, v pravidlech vedle symbolů abecedy popisovaného jazyka, tak zvaných *terminálních symbolů* či stručněji *terminálů*, používáme i "proměnné" neboli *neterminály* (v našem případě E).

Možné *odvození*, neboli *derivace*, slova $a + a \times a$ pak může vypadat takto:

$E \Rightarrow E + E \Rightarrow a + E \Rightarrow a + E \times E \Rightarrow a + a \times E \Rightarrow a + a \times a$

Uvedli jsme příklad tzv. *levé derivace*, kdy jsme v každém kroku přepisovali nejlevější neterminál (či přesněji řečeno nejlevější výskyt neterminálního symbolu). Uveďme příklad pravé derivace pro totéž slovo:

$E \Rightarrow E + E \Rightarrow E + E \times E \Rightarrow E + E \times a \Rightarrow E + a \times a \Rightarrow a + a \times a$

A ještě příklad derivace, která není ani levá ani pravá:

$E \Rightarrow E + E \Rightarrow E + E \times E \Rightarrow E + a \times E \Rightarrow a + a \times E \Rightarrow a + a \times a$

Je zřejmé, že se vlastně ve všech třech případech jedná o jedno a totéž odvození – jen pořadí přepisování neterminálů je různé. Strukturu odvození nezávislou na pořadí přepisování neterminálů zachycuje tzv. *strom odvození*, neboli *derivační strom*. V našem případě je derivační strom odpovídající všem třem derivacím znázorněn na obr. 10 vlevo.

Slovo $a + a \times a$ má ovšem i jinou *levou* derivaci než tu uvedenou výše, a sice:

$E \Rightarrow E \times E \Rightarrow E + E \times E \Rightarrow a + E \times E \Rightarrow a + a \times E \Rightarrow a + a \times a$

Této derivaci odpovídá *jiný* derivační strom – je zachycen na obr. 10 vpravo.

Existence dvou různých derivačních stromů (neboli dvou různých levých derivací) pro jedno slovo jazyka, je nežádoucí vlastnost – příslušná gramatika

(tj. soubor přepisovacích pravidel) je *nejednoznačná* (o tomto problému se ještě zmíníme později).

Naši gramatiku (10.1) lze ovšem nahradit gramatikou

$$\langle EXPR\rangle \longrightarrow \langle EXPR\rangle + \langle TERM\rangle \mid \langle TERM\rangle$$
$$\langle TERM\rangle \longrightarrow \langle TERM\rangle \times \langle FACTOR\rangle \mid \langle FACTOR\rangle$$
$$\langle FACTOR\rangle \longrightarrow (\langle EXPR\rangle) \mid a$$

či stručněji

$$E \longrightarrow E + T \mid T$$
$$T \longrightarrow T \times F \mid F$$
$$F \longrightarrow (E) \mid a$$

která je s původní gramatikou ekvivalentní (tj. popisuje tentýž jazyk) a je přitom jednoznačná. Např. naše slovo $a+a\times a$ má v ní jedinou levou derivaci (jediný derivační strom):

$E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow a + T \Rightarrow a + T \times F \Rightarrow a + F \times F \Rightarrow a + a \times F \Rightarrow a + a \times a$

Naše příklady uvedly tzv. *bezkontextové gramatiky*. Tyto gramatiky reprezentují (generují) tzv. *bezkontextové jazyky*; jejich definici a způsob reprezentace jazyka nyní zformalizujeme.

Definice 10.1 *Bezkontextová gramatika je čtveřice (tj. je dána čtveřicí parametrů)* $G = (\Pi, \Sigma, S, P)$*, kde*

- Π *je konečná množina* neterminálních symbolů *(neterminálů)*
- Σ *je konečná množina* terminálních symbolů *(terminálů), přičemž* $\Pi \cap \Sigma = \emptyset$
- $S \in \Pi$ *je* počáteční *(startovací)* neterminál
- P *je konečná množina* pravidel *typu* $A \to \beta$*, kde*
  - A *je neterminál, tedy* $A \in \Pi$
  - β *je řetězec složený z terminálů a neterminálů, tedy* $\beta \in (\Pi \cup \Sigma)^*$.

Uvažujme lib. $\gamma, \delta \in (\Pi \cup \Sigma)^*$*. Řekneme, že* γ lze přímo přepsat na *(či* přímo odvodí*)* δ *(podle pravidel gramatiky* G*), značíme* $\gamma \Rightarrow_G \delta$ *nebo jen* $\gamma \Rightarrow \delta$ *když* G *zřejmá z kontextu, jestliže existují slova* μ_1, μ_2 *tž.* $\gamma = \mu_1 A \mu_2$*,* $\delta = \mu_1 \beta \mu_2$*, kde* $A \to \beta$ *je pravidlo v* P*.*

Řekneme, že γ lze přepsat na *(*odvodí*)* δ*, značíme* $\gamma \Rightarrow^* \delta$*, jestliže existuje posloupnost* $\mu_0, \mu_1, \ldots, \mu_n$ *slov z* $(\Pi \cup \Sigma)^*$ *(pro nějaké* $n \geq 0$*) tž.* $\gamma = \mu_0 \Rightarrow \mu_1 \Rightarrow \ldots \Rightarrow \mu_n = \delta$*. Zmíněnou posloupnost pak nazveme* odvozením *(*derivací*) délky* n *slova* δ *ze slova* γ*.*

Jazyk generovaný gramatikou G*, označme jej* $L(G)$*, je definován takto:* $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$*.*

Dvě gramatiky G_1, G_2 *nazveme* ekvivalentní*, jestliže* $L(G_1) = L(G_2)$*.*

Jazyk L *je* bezkontextový*, jestliže existuje bezkontextová gramatika* G *taková, že* $L(G) = L$*.*

Poznámky.

- Relace $\Rightarrow^*$ je reflexivním a tranzitivním uzávěrem relace $\Rightarrow$.
- $\gamma \Rightarrow^* \delta$ čteme také 'δ dostaneme z γ', 'γ generuje δ' apod.
- Výše zmíněné *odvození* $\mu_0 \Rightarrow \mu_1 \Rightarrow \ldots \Rightarrow \mu_n$ nazveme *minimální*, jestliže $\mu_i \neq \mu_j$ pro $i \neq j$. Je zřejmé, že jestliže $\gamma \Rightarrow^* \delta$, pak δ lze z γ odvodit (i nějakým) minimálním odvozením. V dalším budeme odvozením automaticky myslet minimální odvození.
- Není-li řečeno jinak, značíme jednotlivé terminály symboly $a, b, c, \ldots$, terminální slova $u, v, w, \ldots$, neterminály $A, B, C, \ldots, X, Y, Z$, řetezce neterminálů a terminálů $\alpha, \beta, \gamma, \ldots$.
- Uvedené příklady již ilustrovaly častý způsob zápisu bezkontextových gramatik, kdy udáváme kompaktně všechny pravé strany pravidel, jež mají týž neterminál na levé straně – tyto pravé strany pak vzájemně oddělujeme svislou čarou "|".

Definice 10.2 *Mějme bezkontextovou gramatiku* $G = (\Pi, \Sigma, S, P)$*; řekneme, že* α *lze přepsat na* β levým přepsáním*, jestliže v* P *ex. pravidlo* $X \to \gamma$ *tž.* $\alpha = uX\delta$*,* $\beta = u\gamma\delta$ *pro nějaké* $u \in \Sigma^*$*,* $\delta \in (\Pi \cup \Sigma)^*$*. Odvození* $\alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n$ *je levým odvozením (levou derivací), jestliže pro vš.* $i = 0, 1, \ldots, n-1$ *lze* α_i *přepsat na* α_{i+1} *levým přepsáním. (Pravé odvození lze definovat obdobně.)*

Lze snadno ukázat, že platí-li $X \Rightarrow_G^* w$, pak w lze z X odvodit (nějakým) levým odvozením i (nějakým) pravým odvozením.

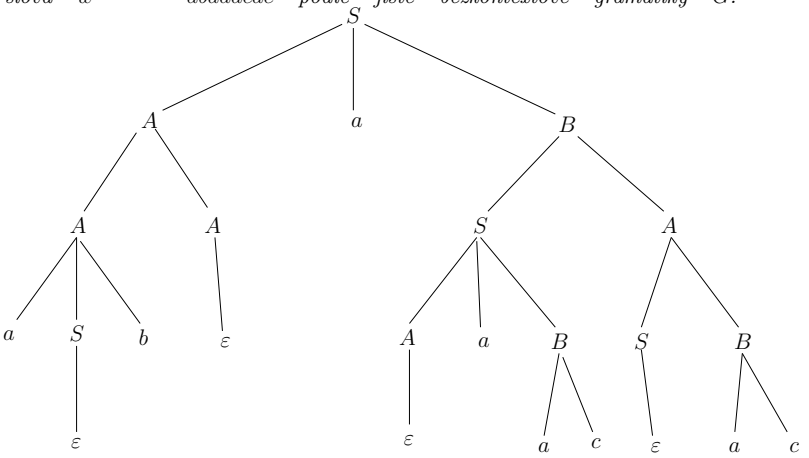
Definice 10.3 *Derivační strom, vztahující se k bezkontextové gramatice* $G = (\Pi, \Sigma, S, P)$*, je uspořádaný kořenový strom (tj. souvislý graf bez cyklů,*

s vyznačeným vrcholem-kořenem, následníci každého vrcholu jsou uspořádáni 'zleva doprava'), jehož vrcholy jsou ohodnoceny symboly z $\Pi \cup \Sigma$*; přitom kořen je ohodnocen symbolem* S *a lib. vrchol s následníky je ohodnocen neterminálem* $X \in \Pi$*, přičemž tito následníci jsou ohodnoceni* $Y_1, Y_2, \ldots, Y_n$ *(*$Y_i \in \Pi \cup \Sigma$*), kde* $X \to Y_1 Y_2 \ldots Y_n$ *je pravidlo v* P*. V případě pravidla* $X \to \varepsilon$ *připouštíme následníka ohodnoceného* ε*. (Brzy uvidíme, že se bez těchto* ε*-pravidel můžeme obejít.)*

Jsou-li listy derivačního stromu zleva doprava ohodnoceny terminály $a_1, a_2, \ldots, a_n$*, říkáme, že se jedná o derivační strom pro slovo* $w = a_1 a_2 \ldots a_n$*.*

Všimněme si, že každému odvození slova w v gramatice G odpovídá (přirozeným způsobem) právě jeden derivační strom pro w; derivačnímu stromu pro w odpovídá obecně více odvození slova w, ovšem např. právě jedno levé odvození.

Úkol 50 *Na obrázku je derivační strom popisující odvození slova* $w = abaaacac$ *podle jisté bezkontextové gramatiky* G*.*



- *Napište levé odvození slova* w *podle gramatiky* G*.*
- *Napište pravé odvození slova* w *podle gramatiky* G*.*
- *Najděte rozklad* $w = w_1 w_2 w_3$ *s* $w_2 \neq \varepsilon$ *tak, aby slovo* $w_1 w_2 w_2 w_3$ *také patřilo do* $L(G)$*.*

Úkol 51 *Navrhněte bezkontextové gramatiky generující následující jazyky:*

- $L_1 = \{ w \in \{a,b\}^* \mid w \text{ obsahuje podslovo } baab \}$
- $L_2 = \{ w \in \{a,b\}^* \mid |w|_b \bmod 3 = 0 \}$
- $L_3 = \{ ww^R \mid w \in \{a,b\}^* \}$
- $L_4 = \{ 0^n 1^m 0^n \mid m,n \geq 0 \}$
- $L_5 = \{ 0^n 1^m \mid 1 \leq n \leq m \leq 2n \}$

Úkol 52 *Snažte se co nejvýstižněji charakterizovat jazyk generovaný gramatikou*
$S \longrightarrow bSS \mid a$

Kapitola 11

Úloha syntaktické analýzy v překladačích

Úlohou syntaktické analýzy v překladačích je rozpoznat, zda zadávaný program (tj. zadávaná posloupnost znaků) je skutečně programem, tj. zda je (syntakticky) správně utvořen. Nestačí ale jen odpověď Ano/Ne. V kladném případě je používaným výstupem např. derivační strom (resp. jeho vhodná reprezentace), jenž slouží jako vstup pro další fáze překladače.

Pro konkrétnější představu se podívejte na 'výsek' z jednoduchého překladače, který je uveden na konci této přednášky. Všimněte si, že derivační (pod)stromy na obr. 50 by vedly k sémanticky (tj. významově) různým cílovým programům !

11.1 Jednoznačné gramatiky a jazyky.

Uvedené úvahy mj. ilustrují, proč je důležitou vlastností gramatik tzv. jednoznačnost:

Definice 11.1 *Řekneme, že bezkont.* gramatika G *je* jednoznačná, *jestliže každé slovo z* $L(G)$ *má právě jedno levé odvození (tj. právě jeden derivační strom). V opačném případě je* G nejednoznačná *(či* víceznačná*).*

Úkol 53 *Lze v úkolu 50 z dostupné informace zjistit něco ohledně víceznačnosti příslušné gramatiky ?*

Viděli jsme, že např. víceznačnou gramatiku (10.1) je možné transformovat na ekvivalentní jednoznačnou gramatiku. Bohužel toto není možné vždy:

Definice 11.2 Bezkontextový jazyk L, *který lze generovat jednoznačnou gramatikou (tj.: ex. jednoznačná bezkontextová gramatika G tž. $L(G) = L$) se nazývá* jednoznačný*; v opačném případě se L nazývá (*vnitřně*)* nejednoznačný.

Např. jazyk $L_1 = \{ a^n b^n \mid n \geq 0 \}$ je generován jednoznačnou bezkontextovou gramatikou

$$S \longrightarrow aSb \mid \varepsilon$$

Jazyk $L_2 = \{ a^i b^j c^k \mid (i = j) \vee (j = k) \}$ generuje např. gramatika

$$S \longrightarrow S_1 C \mid A S_2$$
$$S_1 \longrightarrow a S_1 b \mid \varepsilon$$
$$C \longrightarrow c\, C \mid \varepsilon$$
$$S_2 \longrightarrow b S_2 c \mid \varepsilon$$
$$A \longrightarrow aA \mid \varepsilon$$

Tato gramatika jednoznačná není (proč ?) a dá se dokázat (ne zcela triviálně), že neexistuje jednoznačná bezkontextová gramatika generující L_2; L_2 je tedy nejednoznačný (bezkontextový) jazyk.

11.2 Ilustrace jednoduchého překladu

Omezme se na pascalské přiřazovací příkazy typu $V := E$, kde V je identifikátor proměnné typu real a E je aritmetický výraz vytvořený z identifikátorů proměnných a zápisů čísel typu real pomocí operátorů $+, *$ a pomocí závorek. Takovým příkazem je např.

$$Zisk := (Cena + Dan) * 0.12 \tag{11.1}$$

Všimněme si, že všechny takové příkazy lze generovat bezkontextovou gramatikou G ve tvaru

$$S \longrightarrow \langle id \rangle := E$$
$$E \longrightarrow E + E \mid E * E \mid (E) \mid \langle id \rangle$$

(kde $\{S, E\}$ je množina neterminálů, S počáteční neterminál a $\{:=, +, *, (,), \langle id \rangle\}$ množina terminálů) za předpokladu, že každý výskyt terminálu $\langle id \rangle$ bude nahrazen konkrétním identifikátorem proměnné nebo zápisem čísla typu real.

Chceme navrhnout algoritmus, který libovolný zmíněný pascalský příkaz přeloží do asembleru stroje s jediným pracovním registrem, zvaným akumulátor (ACC), s pamětí tvořenou posloupností (adresovaných) buněk a s následujícím instrukčním repertoárem:

Instrukce	Efekt
LOAD m	$c(m) \rightarrow ACC$
STORE m	$c(ACC) \rightarrow m$
ADD m	$c(ACC) + c(m) \rightarrow ACC$
MPY m	$c(ACC) * c(m) \rightarrow ACC$
LOAD $= m$	$m \rightarrow ACC$
ADD $= m$	$c(ACC) + m \rightarrow ACC$
MPY $= m$	$c(ACC) * m \rightarrow ACC$

K vysvětlení snad stačí následující poznámky:

- např. $c(m) \rightarrow ACC$ znamená, že obsah paměťové buňky m (tedy buňky s adresou m) se zkopíruje do akumulátoru
- výraz $= m$ znamená přímo numerickou hodnotu m
- předpokládáme, že ADD a MPY jsou "floating-point" operace

Práce překladače se dá rozdělit zhruba do následujících fází:

- lexikální analýza, kdy se ve zpracovávaném zdrojovém textu (tj. ve vstupním řetězci) zjistí tzv. lexikální jednotky (např. identifikátory, zápisy čísel, znaky $+, *, :=$ apod.),
- syntaktická analýza, kdy se zjistí syntaktická struktura řetězce předzpracovaného lexikální analýzou,
- generování kódu, kdy se s využitím zjištěné syntaktické struktury vytváří cílový kód (tj. překlad vstupního řetězce).

(V reálném případě se tyto fáze různě prolínají, jsou doplněny o další fáze – např. o optimalizaci kódu, zotavení z chyb apod.; ale to není pro náš příklad podstatné).

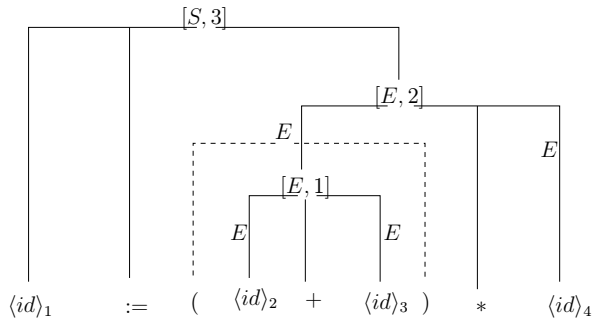
Výsledkem lexikální analýzy vstupního řetězce (11.1) by mohl být řetězec

$$\langle id\rangle_1 := (\langle id\rangle_2 + \langle id\rangle_3) * \langle id\rangle_4 \qquad (11.2)$$

zároveň s tabulkou TAB:

Poř. číslo	Identifikátor	Informace
1	Zisk	prom. real
2	Cena	prom. real
3	Dan	prom. real
4	0.12	konst. real

Výsledkem syntaktické analýzy pro řetězec (11.2) by mohl být (derivační) strom na obrázku 11.1, ve kterém číslo v ohodnocení vnitřních uzlů udává maximální vzdálenost k listu.



Obrázek 11.1: Výstup syntaktické analýzy

(Derivační strom podle G by měl 7 vnitřních uzlů. Zde předpokládáme, že "zbytečné" uzly byly vyhozeny, takže výsledný strom má jen 3 vnitřní uzly; např. závorky jsou potřeba k správnému vytvoření stromu, ale pro další účely nejsou potřebné).

Výsledný kód lze ze sestrojeného stromu sestavit pomocí následujících pravidel, která každému vnitřnímu uzlu u přiřazují $Cod(u)$:

- uzel u je ohodnocen $\langle id\rangle_i$: jestliže i-tá položka v tabulce TAB je proměnná typu real, pak $Cod(u)$ je příslušný identifikátor (např. je-li u ohodnocen $\langle id\rangle_1$, je $Cod(u)$ řetězec 'Zisk'); jestliže i-tá položka v tabulce TAB je konstanta (typu real), pak $Cod(u)$ je příslušný zápis čísla předcházený znakem = (např. je-li u ohodnocen $\langle id\rangle_4$, je $Cod(u)$ řetězec '= 0.12')

- je-li uzel u ohodnocen některým ze symbolů $:=, +, *$ pak $Cod(u)$ je prázdný řetězec
- uzel u je ohodnocen $[S, n]$ a má následníky u_1, u_2, u_3: Cod(u) je pak

  'LOAD' $Cod(u_3)$ '; STORE' $Cod(u_1)$

- uzel u je ohodnocen $[E, n]$ a má následníky u_1, u_2, u_3: pak
  - jestliže u_2 je ohodnocen $+$, $Cod(u)$ je

    $Cod(u_3)$ '; STORE \$'n '; LOAD' $Cod(u_1)$ '; ADD \$'n
  - jestliže u_2 je ohodnocen $*$, $Cod(u)$ je

    $Cod(u_3)$ '; STORE \$'n '; LOAD' $Cod(u_1)$ '; MPY \$'n

Např. kód příslušný k uzlu ohodnocenému $[E, 1]$ je

Dan ; STORE \$1 ; LOAD $Cena$; ADD \$1

Kód příslušný k uzlu ohodnocenému $[S, 3]$ (tedy kýžený program v asembleru) je

LOAD = 0.12 ;
STORE \$2 ;
LOAD Dan ;
STORE \$1 ;
LOAD $Cena$;
ADD \$1 ;
MPY \$2 ;
STORE $Zisk$

Kapitola 12

Speciální formy bezkontextových gramatik

Přednáška 9

12.1 Redukované gramatiky

Vzpomeňme si na odstraňování nedosažitelných stavů u konečných automatů – takové stavy jsou "zbytečné". Teď se podíváme na odstraňování "zbytečných" neterminálů u bezkontextových gramatik. Neterminál je zbytečný, jestliže z něj nelze odvodit žádné terminální slovo (pak se tedy nemůže objevit v žádném odvození terminálního slova z počátečního neterminálu), nebo je nedosažitelný – nemůže se prostě vůbec objevit při jakémkoli přepisování začínajícím z počátečního neterminálu. Redukovaná gramatika neobsahuje takové zbytečné neterminály:

Definice 12.1 *Bezkontextová gramatika $G = (\Pi, \Sigma, S, P)$ se nazývá redukovaná, jestliže jsou pro každý $X \in \Pi$ splněny tyto dvě podmínky:*

1. *existuje aspoň jedno $w \in \Sigma^*$ tž. $X \Rightarrow^* w$,*
2. *existují slova $\alpha, \beta \in (\Pi \cup \Sigma)^*$ tž. $S \Rightarrow^* \alpha X \beta$.*

Uvažujme nejprve, jak pro danou gramatiku $G = (\Pi, \Sigma, S, P)$ zjistit neterminály splňující podmínku 1.

Chceme tedy zkonstruovat množinu $\mathcal{T} = \{X \in \Pi \mid \exists w \in \Sigma^* : X \Rightarrow^* w\}$. Konstruujeme postupně množiny $\mathcal{T}_1, \mathcal{T}_2, \ldots$, kde $\mathcal{T}_1 = \{X \in \Pi \mid \exists w \in \Sigma^* : (X \to w) \in P\}$ a $\mathcal{T}_{i+1} = \mathcal{T}_i \cup \{X \in \Pi \mid \exists \alpha \in \mathcal{T}_i^* : (X \to \alpha) \in P\}$, až k případu $\mathcal{T}_n = \mathcal{T}_{n+1}$. Na takový případ nutně narazíme pro $n \leq |\Pi|$ a očividně platí $\mathcal{T}_n = \mathcal{T}$.

Neterminály, slňující podmínku 2., tedy dosažitelné neterminály, lze zjistit takto:

Množinu $\mathcal{D} = \{X \in \Pi \mid \exists \alpha, \beta : S \Rightarrow_G^* \alpha X \beta\}$ sestrojíme zase postupnou konstrukcí $\mathcal{D}_1, \mathcal{D}_2, \ldots$, kde $\mathcal{D}_1 = \{S\}$ a $\mathcal{D}_{i+1} = \mathcal{D}_i \cup \{X \in \Pi \mid$ ex. $Y \in \mathcal{D}_i$ a α obsahující X tž. $(Y \to \alpha) \in P\}$, až k případu $\mathcal{D}_n = \mathcal{D}_{n+1}$.

Snadno teď ukážeme:

Věta 12.1.1 *Ke každé bezkontextové gramatice G tž. $L(G) \neq \emptyset$ lze sestrojit ekvivalentní redukovanou gramatiku.*

Důkaz. Mějme $G = (\Pi, \Sigma, S, P)$. Nejdříve zkonstruujeme množinu neterminálů splňujících podmínku 1. (z definice redukované gramatiky).

Pak v G vynecháme všechny neterminály nesplňující 1. a všechna pravidla, která takové neterminály obsahují. Dostaneme tak jistou gramatiku G' a je očividné, že $L(G) = L(G')$.

Pro gramatiku G' nyní zkonstruujeme množinu neterminálů splňujících podmínku 2. a dále vynecháme všechny neterminály nesplňující 2. a všechna pravidla, která takové neterminály obsahují. Dostaneme tak jistou gramatiku G'' a je opět očividné, že $L(G) = L(G') = L(G'')$.

Přesvědčte se, že G'' je skutečně redukovanou gramatikou.
Konec Důkazu

Úkol 54 *Zredukujte následující bezkontextové gramatiky:*

$S \longrightarrow aSb \,|\, aAbb \,|\, \varepsilon$
$A \longrightarrow aAB \,|\, bB$
$B \longrightarrow aAb \,|\, BB$
$C \longrightarrow CC \,|\, cS$

$S \longrightarrow aA \,|\, bB \,|\, aSa \,|\, bSb \,|\, \varepsilon$
$A \longrightarrow bCD \,|\, Dba$
$B \longrightarrow Bb \,|\, AC$
$C \longrightarrow aA \,|\, c$
$D \longrightarrow DE$
$E \longrightarrow \varepsilon$

Úkol 55 *Přehození uvedeného postupu (tj. nejprve odstranění neterminálů nesplňujících 2. a pak těch nesplňujících 1. nemusí vést k redukované gramatice.*

Snadno teď také můžeme ukázat tuto větu:

Věta 12.1.2 *Existuje algoritmus, který pro libovolnou bezkontextovou gramatiku G rozhodne, zda $L(G) = \emptyset$.*

Důkaz. Stačí ověřit, zda S patří do množiny neterminálů splňujících podmínku 1.
Konec Důkazu

Úkol 56 *Zjistěte, zda pro následující gramatiku G je $L(G) \neq \emptyset$*
$S \longrightarrow aS \,|\, AB \,|\, CD$
$A \longrightarrow aDb \,|\, AD \,|\, BC$
$B \longrightarrow bSb \,|\, BB$
$C \longrightarrow BA \,|\, ASb$
$D \longrightarrow ABCD \,|\, \varepsilon$

Dále poznamenejme, že na rozdíl od konečných automatů neexistuje algoritmus, který by k dané bezkontextové gramatice zkonstruoval nejmenší s ní ekvivalentní. Dá se to ukázat metodami teorie vyčíslitelnosti, z nichž rovněž plyne, že neexistuje algoritmus, který by rozhodoval ekvivalenci bezkontextových gramatik. (To bude demonstrováno v kursu o vyčíslitelnosti a složitosti.)

12.2 Nevypouštějící gramatiky

Již jsme dříve zmínili možnost zbavení se (z technických důvodů nepříjemných) tzv. ε-pravidel (pravidel typu $X \longrightarrow \varepsilon$):

Definice 12.2 *Bezkontextová gramatika se nazývá nevypouštějící , jestliže neobsahuje žádné pravidlo typu $X \longrightarrow \varepsilon$.*

Věta 12.2.1 *Ke každé bezkontextové gramatice G lze sestrojit ekvivalentní nevypouštějící gramatiku G' tž. $L(G') = L(G) - \{\varepsilon\}$.*

Důkaz. Konstrukce využívá obdoby výše uvedené konstrukce pro neterminály splňující podmínku 1. z definice redukované gramatiky.

Ke gramatice $G = (\Pi, \Sigma, S, P)$ totiž nejprve sestrojíme množinu $\mathcal{E} = \{X \in \Pi \mid X \Rightarrow^* \varepsilon\}$; zde opět konstruujeme množiny $\mathcal{E}_1, \mathcal{E}_2, \ldots$, kde $\mathcal{E}_1 = \{X \in \Pi \mid (X \to \varepsilon) \in P\}$ a $\mathcal{E}_{i+1} = \mathcal{E}_i \cup \{X \in \Pi \mid \exists \alpha \in \mathcal{E}_i^* : (X \to \alpha) \in P\}$. Skončíme v případě $\mathcal{E}_n = \mathcal{E}_{n+1}$ – je zřejmé, že pak $\mathcal{E}_n = \mathcal{E}$.

Na základě $\mathcal{E}$ sestrojíme množinu pravidel P' takto: pro každé pravidlo $(X \to \alpha) \in P$ zařadíme do P' všechna možná pravidla $X \to \beta$, kde β vznikne z α vypuštěním některých (třeba žádných) výskytů symbolů z $\mathcal{E}$; přitom ovšem vynecháme (nezařazujeme) případnou možnost $X \to \varepsilon$.

Položíme $G' = (\Pi, \Sigma, S, P')$; lze snadno ověřit, že skutečně $L(G') = L(G) - \{\varepsilon\}$ (formálně lze postupovat např. indukcí podle délky odvození).
Konec Důkazu

Důsledek 12.2.2 *Ke každé bezkontextové gramatice $G = (\Pi, \Sigma, S, P)$ existuje ekvivalentní bezkontextová gramatika $G_1 = (\Pi_1, \Sigma, S_1, P_1)$, kde ε může být pravou stranou pouze u pravidla $S_1 \to \varepsilon$; v takovém případě se pak S_1 nevyskytuje na pravé straně žádného z pravidel z P_1.*

Důkaz. Ke G lze sestrojit nevypouštějící gramatiku $G' = (\Pi, \Sigma, S, P')$. Platí-li $\varepsilon \notin L(G)$ (S nepatří do výše zmíněné $\mathcal{E}$), položíme $G_1 = G'$. Je-li $\varepsilon \in L(G)$, vznikne G_1 z G' přidáním nového neterminálu S_1, který bude počátečním, a pravidel $S_1 \to \varepsilon$, $S_1 \to S$.
Konec Důkazu

Úkol 57 *K bezkontextové gramatice G dané uvedenými pravidly sestrojte nevypouštějící gramatiku G' takovou, že $L(G') = L(G) - \{\varepsilon\}$.*
$S \longrightarrow AB \,|\, \varepsilon$
$A \longrightarrow aAAb \,|\, BS \,|\, CA$
$B \longrightarrow BbA \,|\, CaC \,|\, \varepsilon$
$C \longrightarrow aBB \,|\, bS$

12.3 Chomského normální forma

Z technických důvodů je užitečné, že bezkontextové gramatiky lze transformovat do různých *normálních forem*, u nichž jsou kladena další syntaktická omezení na povolená pravidla. Příkladem je tzv. Chomského normální forma:

Definice 12.3 *Bezkontextová gramatika je v Chomského normální formě, zkráceně v CHNF, jestliže každé její pravidlo je tvaru $X \to YZ$ nebo $X \to a$, kde X, Y, Z označují neterminální symboly a a terminální symbol.*

Věta 12.3.1 *Ke každé bezkontextové gramatice G lze sestrojit gramatiku G' v CHNF tž. $L(G') = L(G) - \{\varepsilon\}$.*

Důkaz. Podle věty 12.2.1 můžeme rovnou předpokládat, že G je nevypouštějící (jinak ji do této formy převedeme). Ukážeme, jak postupnou transformací G zkonstruujeme ekvivalentní gramatiku G' v CHNF.

Nejprve z gramatiky G odstraníme pravidla typu $X \to Y$:

Pro každý neterminál A zkonstruujeme množinu $\mathcal{D}_A = \{B \mid A \Rightarrow^* B\}$ (jak ?); pak pro každé pravidlo $B \to \alpha$, kde $B \in \mathcal{D}_A$ a α není rovno jednomu neterminálu, přidáme pravidlo $A \to \alpha$. Nakonec odstraníme všechna pravidla typu $X \to Y$. Snadno lze ověřit, že generovaný jazyk zůstává zachován.

Dále pro každý terminál a zavedeme nový neterminál A_a a přidáme pravidlo $A_a \to a$. Pak na pravé straně každého pravidla $X \to \alpha$, kde $|\alpha| \geq 2$, nahradíme každý výskyt terminálu a neterminálem A_a.

Je zřejmé, že generovaný jazyk zůstává stále zachován; jediná pravidla porušující podmínku CHNF mohou být typu $X \to Y_1 Y_2 \dots Y_n$, kde $n \geq 3$ (Y_i jsou samozřejmě neterminály).

Každé pravidlo uvedeného typu lze ovšem nahradit soustavou $X \to Y_1 Z_1$, $Z_1 \to Y_2 Z_2$, …, $Z_{n-3} \to Y_{n-2} Z_{n-2}$, $Z_{n-2} \to Y_{n-1} Y_n$, kde $Z_1, Z_2, \dots, Z_{n-2}$ jsou nově přidané neterminály.

Opět je snadné se přesvědčit, že generovaný jazyk se nezmění a uvedenými změnami vzniklá gramatika G' je požadovanou gramatikou v CHNF.
Konec Důkazu

Úkol 58 *Následující gramatiky převeďte do Chomského normální formy:*

$S \longrightarrow A \,|\, 0SA \,|\, \varepsilon$
$A \longrightarrow 1A \,|\, 1 \,|\, B1$
$B \longrightarrow 0B \,|\, 0 \,|\, \varepsilon$
$S \longrightarrow abS \,|\, CaS \,|\, BaS \,|\, a$
$B \longrightarrow aCB \,|\, SC$
$C \longrightarrow BCb \,|\, SB$

$S \longrightarrow (E)$
$E \longrightarrow F + F \,|\, F \times F$
$F \longrightarrow a \,|\, S$

12.4 Greibachové normální forma

V některých situacích je užitečná tato normální forma:

Definice 12.4 *Bezkontextová gramatika je v Greibachové normální formě, zkráceně v GNF, jestliže každé její pravidlo je v tvaru $X \to aY_1Y_2 \dots Y_n$ ($n \geq 0$, a je terminál, Y_i jsou neterminály).*

Věta 12.4.1 *Ke každé bezkontextové gramatice G lze sestrojit gramatiku G' v GNF tž. $L(G') = L(G) - \{\varepsilon\}$.*

Podrobný důkaz zde neuvádíme; zmiňme ale, že základní ‘procedury’ při převodu gramatiky do GNF jsou zachyceny v následujících dvou lemmatech. První je očividné:

Lemma 12.4.2 *Mějme bezkont. gramatiku $G = (\Pi, \Sigma, S, P)$. Nechť P obsahuje pravidlo $A \to \alpha B \gamma$ a $B \to \beta_1$, $B \to \beta_2$, …, $B \to \beta_n$ jsou všechna pravidla s B na levé straně. Potom odstraníme-li z P pravidlo $A \longrightarrow \alpha B \gamma$ a naopak přidáme pravidla $A \longrightarrow \alpha\beta_1\gamma$, $A \longrightarrow \alpha\beta_2\gamma$, …, $A \longrightarrow \alpha\beta_n\gamma$, dostaneme gramatiku ekvivalentní s G.*

Další lemma je základem pro odstranění *levé rekurze*, tj. případu $X \Rightarrow^* X\alpha$; to je důležité pro syntaktickou analýzu v překladačích.

Lemma 12.4.3 *Mějme bezkont. gramatiku $G = (\Pi, \Sigma, S, P)$. Nechť $A \to A\alpha_1$, $A \to A\alpha_2$, …, $A \to A\alpha_m$, $A \to \beta_1$, $A \to \beta_2$, …, $A \to \beta_n$ jsou všechna pravidla s A na levé straně, přičemž řetězce β_i nezačínají A. Gramatika $G' = (\Pi \cup \{Z\}, \Sigma, S, P')$ vzniklá z G dodáním nového neterminálu Z a nahrazením všech uvedených pravidel soustavou $A \to \beta_i$, $A \to \beta_i Z$ ($i = 1, 2, \dots, n$), $Z \to \alpha_i$, $Z \to \alpha_i Z$ ($i = 1, 2, \dots, m$), je ekvivalentní gramatice G.*

Kapitola 13

Zásobníkové automaty; ekvivalence s bezkontextovými gramatikami

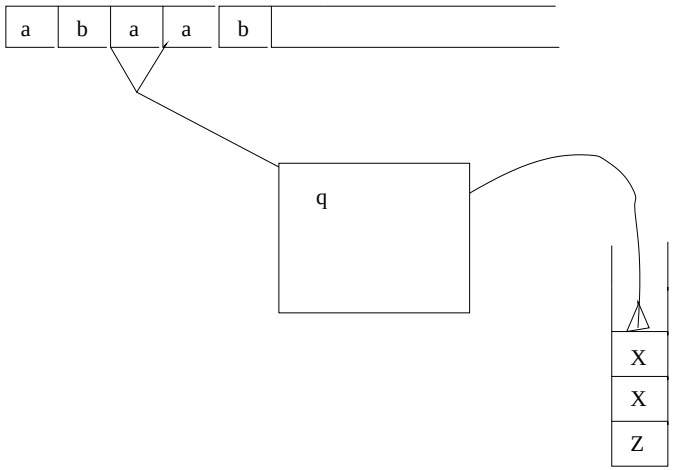
Přednáška 10

Víme, že např. jazyk

$\{\langle begin\rangle\langle end\rangle, \langle begin\rangle\langle begin\rangle\langle end\rangle\langle end\rangle, \langle begin\rangle\langle begin\rangle\langle begin\rangle\langle end\rangle\langle end\rangle\langle end\rangle, \dots\}$

nebo "ekvivalentní" jazyk $L = \{a^n b^n \mid n \geq 1\}$ nelze rozpoznávat konečným automatem.

Snadno ovšem takový jazyk (tedy slova daného jazyka) rozpoznáme zařízením podobným konečnému automatu, které může navíc používat neomezenou (tedy potenciálně nekonečnou) paměť typu *zásobník*: přečtené symboly a se prostě ukládají do zásobníku a při čtení symbolů b se pak tyto zásobníkové symboly odebírají – tím způsobem jsme schopni počet a-ček a b-ček porovnat. Zmíněnému zařízení budeme říkat zásobníkový automat, zkráceně ZA. "Vnější pohled" na ZA je ilustrován obrázkem 13.



Čtenář by si měl být schopen udělat představu, jak takové zařízení pracuje a jakým způsobem reprezentuje (rozpoznává) jazyk. Tuto představu je pak potřebné konfrontovat s níže uvedenou definicí (která odstraňuje všechny případné nejasnosti či nejednoznačnosti). Zdůrazněme hned, že obecným termínem "zásobníkový automat" se obvykle myslí "*nedeterministický* zásobníkový automat".

Definice 13.1 *Zásobníkový automat, zkráceně (ZA) M je dán parametry $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$, kde*

- Q *je konečná neprázdná množina* stavů
- Σ *je konečná neprázdná množina* vstupních symbolů *(vstupní abeceda)*
- Γ *je konečná neprázdná množina* zásobníkových symbolů *(zásobníková abeceda)*
- $q_0 \in Q$ *je* počáteční stav
- $Z_0 \in \Gamma$ *je* počáteční zásobníkový symbol
- δ *je zobrazení množiny* $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$ *do množiny všech konečných podmnožin množiny* $Q \times \Gamma^*$.

Konfigurací zásobníkového automatu $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ *rozumíme trojici* (q, w, α), *kde* $q \in Q$, $w \in \Sigma^*$, $\alpha \in \Gamma^*$.

Na množině všech konfigurací automatu M *definujeme relaci* $\vdash_M$*:*

$(q, aw, X\beta) \vdash_M (q', w, \alpha\beta) \iff_{df} \delta(q, a, X) \ni (q', \alpha)$

kde $a \in (\Sigma \cup \{\varepsilon\})$, $w \in \Sigma^*$, $\beta \in \Gamma^*$. *Říkáme pak, že konfigurace* $(q, aw, X\beta)$ bezprostředně vede ke *(resp. může bezprostředně vést ke) konfiguraci* $(q', w, \alpha\beta)$ *apod.*

Relace $\vdash_M^*$ *je reflexivním a tranzitivním uzávěrem relace* $\vdash_M$. $K_1 \vdash_M^* K_2$ *pak čteme: konfigurace* K_1 *vede k (resp. může vést k)* K_2 *apod.*

Slovo $w \in \Sigma^*$ je přijímáno *ZA* M, *jestliže* $(q_0, w, Z_0) \vdash_M^* (q, \varepsilon, \varepsilon)$ *pro nějaké* $q \in Q$.

Jazykem rozpoznávaným *ZA* M *rozumíme jazyk* $L(M) = \{\, w \in \Sigma^* \mid w$ *je přijímáno ZA* $M \,\}$.

Úkol 59 *Sestrojte zásobníkový automat rozpoznávající jazyk* $L = \{\, wc(w)^R \mid w \in \{a, b\}^* \,\}$.
Pak navrhněte ZA rozpoznávající jazyk $\{ww^R \mid w \in \{0, 1\}^*\}$. *Jistě přitom využijete nedeterminismus (o důvodu se zmíníme později).*
Sestrojte ještě zásobníkový automat rozpoznávající jazyk $L = \{\, u \in \{a, b, c\}^* \mid$ *po vynechání všech výskytů symbolu* c *z* u *dostaneme slovo ve tvaru* $w(w)^R \,\}$.

Zásobníkové automaty tvoří "automatový protějšek" k bezkontextovým gramatikám, tj. rozpoznávají právě bezkontextové jazyky. Toto si teď ukážeme. Nejdříve ve směru od gramatice k automatu, což mj. ilustruje základní ideu syntaktické analýzy v překladačích.

Lemma 13.0.4 *Ke každé bezkontextové gramatice* G *lze sestrojit ZA* M *(s jedním stavem) tž.* $L(M) = L(G)$.

Důkaz. Mějme $G = (\Pi, \Sigma, S, P)$. K ní zkonstruujme ZA $M = (\{q_0\}, \Sigma, \Pi \cup \Sigma, \delta, q_0, S)$, kde pro každé $X \in \Pi$ je $\delta(q_0, \varepsilon, X) = \{(q_0, \alpha) \mid (X \to \alpha) \in P\}$ a pro každé $a \in \Sigma$ je $\delta(q_0, a, a) = \{(q_0, \varepsilon)\}$; jiným argumentům přiřazuje δ prázdnou množinu.

Dá se snadno ukázat $S \Rightarrow_G^* u\alpha \iff (q_0, u, S) \vdash_M^* (q_0, \varepsilon, \alpha)$. Zde $u \in \Sigma^*$, $\alpha \in \Pi(\Pi \cup \Sigma)^*$ nebo $\alpha = \varepsilon$; symbolem $\Rightarrow_G^*$ přitom zde označujeme relaci odpovídající *levému* odvození.
Konec Důkazu

ZA uvedený v důkazu provádí (nedeterministicky) tzv. analýzu "shora dolů": sledujíce jistou levou derivaci, snaží se de facto budovat derivační strom pro dané vstupní slovo od kořene k listům (průchodem zleva doprava).

Úkol 60 *Demonstrujte úspěšný běh (nedeterministického) zásobníkového automatu při syntaktické analýze shora dolů slova* $a * (a + a)$ *podle gramatiky*
1/ $A \longrightarrow A + B$
2/ $A \longrightarrow B$
3/ $B \longrightarrow B * C$
4/ $B \longrightarrow C$
5/ $C \longrightarrow (A)$
6/ $C \longrightarrow a$

Úkol 61 *Zkuste se alespoň zamyslet nad konstrukcí ZA ke gramatice (na uvedeném konkrétním příkladu i obecně) tak, aby prováděl analýzu "zdola nahoru", tj., aby sledoval jistou pravou derivaci pozpátku, budujíce derivační strom od listů ke kořeni.*

Na deterministických verzích takových zásobníkových automatů (pro speciální třídy gramatik), jsou založeny algoritmy používané u syntaktické analýzy v reálných překladačích. (Např. se jedná o tzv. LL- či LR-analyzátory.)

Ukázali jsme tedy, že k bezkontextové gramatice lze zkonstruovat ekvivalentní (dokonce jednostavový) zásobníkový automat. V případě jednostavového ZA lze snadno provést i opačnou transformaci, zachycenou následujícím lemmatem.

Lemma 13.0.5 *Ke každému ZA* M *s jedním stavem lze sestrojit bezkontextovou gramatiku* G *tž.* $L(G) = L(M)$.

Důkaz. Mějme $M = (\{q_0\}, \Sigma, \Gamma, \delta, q_0, Z_0)$; předpokládejme $\Sigma \cap \Gamma = \emptyset$ (toho docílíme případným přejmenováním zásobníkových symbolů). Ověřte (viz další Úkol), že následující gramatika je onou požadovanou: $G = (\Gamma, \Sigma, Z_0, P)$, kde $\delta(q_0, a, A) \ni (q_0, \alpha) \iff (A \to a\alpha) \in P$ $(a \in (\Sigma \cup \{\varepsilon\}))$.
Konec Důkazu

Úkol 62 *Uvažujme ZA* $M = (\{q_0\}, \Sigma, \Gamma, \delta, q_0, Z_0)$, *kde* $\Sigma \cap \Gamma = \emptyset$ *a k němu sestrojenou BG* $G = (\Gamma, \Sigma, Z_0, P)$ *takovou, že* $(A \to a\alpha) \in P \iff \delta(q_0, a, A) \ni (q_0, \alpha)$ *(*$a \in (\Sigma \cup \{\varepsilon\})$*.*

Ukažte indukcí (podle počtu kroků odvození), že

$Z_0 \Rightarrow_G^* u\alpha \iff (q_0, u, Z_0) \vdash_M^* (q_0, \varepsilon, \alpha)$ *(zde* $u \in \Sigma^*$, $\alpha \in \Gamma^*$ *a* $\Rightarrow_G^*$ *označuje* levé *odvození).*

Další lemma pak ukáže, že obecný ZA lze převést na jednostavový. To je technicky obtížnější, byť idea není nijak složitá – informaci o řídicím stavu původního ZA M musí mít nový jednostavový (tedy de facto "bezstavový", jakoby s pamětí 0 bitů) ZA M' při "simulaci" původního M vhodně uloženu na zásobníku. Jako obvykle je to "něco za něco": za zrušení řídicích stavů platíme rozšířením zásobníkové abecedy. (Při čtení důkazu je vhodné rovnou provádět úkol 63.)

Lemma 13.0.6 *Ke každému ZA M lze sestrojit ZA M' s jedním stavem tž. $L(M) = L(M')$.*

Důkaz. Idea: Jednostavový ZA M' (stav označíme s) bude mít zásobníkové symboly typu $\langle p, X, q\rangle$, kde p, q jsou stavy a X je zásobníkový symbol automatu M, přičemž bude platit:

$$\forall w : (s, w, \langle p, X, q\rangle) \vdash^*_{M'} (s, \varepsilon, \varepsilon) \iff (p, w, X) \vdash^*_M (q, \varepsilon, \varepsilon)$$

Konkrétně pro $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ konstruujeme $M' = (\{s\}, \Sigma, \Gamma', \delta', s, R)$, kde $\Gamma' = (Q \times \Gamma \times Q) \cup \{R\}$ a δ' je určena následovně:

- $\delta'(s, \varepsilon, R) = \{(s, \langle q_0, Z_0, q\rangle) \mid q \in Q\}$,
- pro $(q', \varepsilon) \in \delta(q, a, X)$ $(a \in (\Sigma \cup \{\varepsilon\}))$ zařadíme do $\delta'(s, a, \langle q, X, q'\rangle)$ prvek (s, ε),
- pro $(q', A_1A_2 \dots A_n) \in \delta(q, a, X)$ $(n \geq 1)$ zařadíme do $\delta'(s, a, \langle q, X, \overline{q}\rangle)$ prvek $(s, \langle q', A_1, q_1\rangle\langle q_1, A_2, q_2\rangle \dots \langle q_{n-1}, A_n, \overline{q}\rangle)$ pro každé $\overline{q}, q_1, q_2, \dots, q_{n-1} \in Q$.

(Chápeme-li δ' jako množinu 'instrukcí', pak lze říci, že δ' je minimální množina instrukcí splňující výše uvedené podmínky.)

Dá se ověřit, že každému přijímajícímu výpočtu automatu M nad slovem w odpovídá přijímající výpočet automatu M' nad w a naopak.
Konec Důkazu

Úkol 63 *K zásobníkovému automatu M se vstupní abecedou $\{a, b\}$, zásobníkovou abecedou $\{A, B\}$, počátečním zásobníkovým symbolem A, množinou stavů $\{p, q, r\}$, počátečním stavem p a přechodovou funkcí δ definovanou následovně*

$\delta(p, a, A) = \{(q, AA), (p, B)\}$,
$\delta(q, b, A) = \{(q, AA)\}$,

$\delta(p, \varepsilon, B) = \{(q, A)\}$,
$\delta(q, \varepsilon, A) = \{(r, \varepsilon)\}$,
$\delta(r, a, A) = \{(r, A)\}$,
$\delta(r, b, A) = \{(r, \varepsilon)\}$

(pro ostatní prvky def. oboru je funkční hodnota rovna $\emptyset$) sestrojte nejdříve jednostavový *ZA rozpoznávající jazyk $L(M)$, a poté gramatiku generující tento jazyk. Použijte přitom konstrukce obsažené ve výše uvedených důkazech.*

Z uvedených lemmat ihned plyne

Věta 13.0.7 *(Nedeterministické) zásobníkové automaty rozpoznávají právě bezkontextové jazyky (a jsou takto ekvivalentní bezkontextovým gramatikám).*

Definovali jsme, že slovo je přijímáno ZA právě tehdy, když se po jeho přečtení ZA může ocitnout v situaci (konfiguraci) s prázdným zásobníkem. Obvykle se uvažuje také forma přijímání slova možným dosažením koncového stavu (řídicí jednotky). Obě alternativy jsou definovány níže a je ukázáno, že obě také mají tutéž rozpoznávací sílu (v případě *nedeterministických* ZA).

Definice 13.2 *Pro ZA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ (kde jsme přidali parametr F, což je množina* koncových (přijímajících) *stavů – $F \subseteq Q$) definujeme* jazyk rozpoznávaný koncovým stavem $L_{KS}(M) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^*_M (q, \varepsilon, \alpha)$ *pro nějaké $q \in F$ a $\alpha \in \Gamma^*\}$ a* jazyk rozpoznávaný prázdným zásobníkem $L_{PZ}(M) = \{w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^*_M (q, \varepsilon, \varepsilon)$ *pro nějaké $q \in Q\}$.*

Lemma 13.0.8 *K libovolnému ZA M_1 lze zkonstruovat ZA M_2 tž. $L_{KS}(M_1) = L_{PZ}(M_2)$ a také M_2' tž. $L_{PZ}(M_1) = L_{KS}(M_2')$.*

Důkaz. Neformální idea spočívá v následujícím: každý ZA lze jednoduše upravit tak, že dodáme nový počáteční zásobníkový symbol B (bottom=dno), který se bude stále vyskytovat na dně zásobníku (a pouze tam) – promyslete si technické podrobnosti ! Pak už je důkaz tvrzení přímočarý.
Konec Důkazu

Kapitola 14

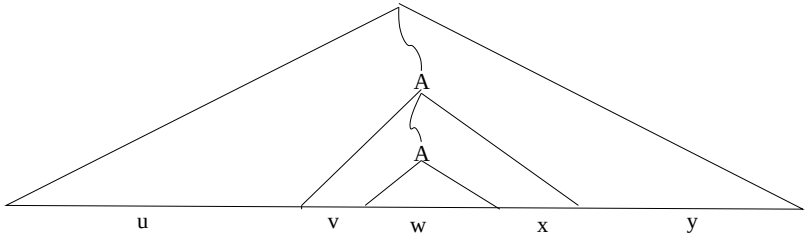
Pumping lemma pro bezkontextové jazyky

Přednáška 11

Připomeňme si, jak jsme dokazovali, že jazyk $\{a^n b^n \mid n \geq 0\}$ není regulární, a jak jsme odvodili pumping lemma platné obecně pro regulární jazyky. Je jasné, že uvedený jazyk přijímá jednoduchý zásobníkový automat. Uvažujme nyní ale jazyk

$$L = \{a^n b^n c^n \mid n \geq 0\}$$

Jistě nás naše intuice brzy dovede k závěru, že na takový jazyk zásobníkový automat (byť nedeterministický) nestačí. Jak to ale jasně dokázat ? Opět přivedením předpokladu, že L je bezkontextový, k logickému sporu. Předpokládejme tedy, že L je bezkontextový a uvažujme bezkontextovou gramatiku G, která ho generuje (uvažovat gramatiku se ukáže pro naše účely vhodnější než uvažovat zásobníkový automat). Pro každé slovo $a^n b^n c^n$ tedy existuje derivační strom (podle gramatiky G). Vezmeme-li slovo "velmi dlouhé" (tj. n "velmi velké"), v příslušném derivačním stromu nutně dochází k opakování nějakého neterminálu na nějaké větvi (tj. cestě od kořene k listu) – viz obr. 14. Přesněji řečeno: derivačních stromů, ve kterých se takové opakování nevyskytuje, je konečně mnoho (proč ?); výraz "n je velmi velké" lze zpřesnit tak, že $3n$ (tj. délka slova $a^n b^n c^n$) je větší než délka nejdelšího slova odvoditelného derivačním stromem bez opakování.



Vezměme nyní tedy ono velmi dlouhé slovo $a^n b^n c^n$ a pro něj *nejmenší* možný derivační strom (i ten má opakování jako na obr. 14). Slovo $a^n b^n c^n$ se dá psát ve tvaru $uvwxy$ (jak znázorněno na obrázku), kde navíc aspoň jedno ze slov v, x je neprázdné (jinak bychom mohli oba uzly označené neterminálem A ztotožnit a získali bychom pro $a^n b^n c^n$ menší derivační strom !). Je jasné, že i pro uwy, a také uv^2wx^2y, uv^3wx^3y, ... existují derivační stromy (viz obr. 14); tato slova tudíž také patří do L.

Snadno se ale můžeme přesvědčit, že ať rozdělíme slovo $a^n b^n c^n$ na 5 částí $uvwxy$ *jakkoliv*, přičemž alespoň jedno ze slov v, x je neprázdné, pak slovo uwy zaručeně nepatří do L (proč ?).

Opět jsme odvodili určité "pumping lemma" platné obecně pro bezkontextové jazyky (nikoli jen pro náš L) a demonstrovali jsme jeho použití pro důkaz, že L není bezkontextový. Zmíněné lemma následuje.

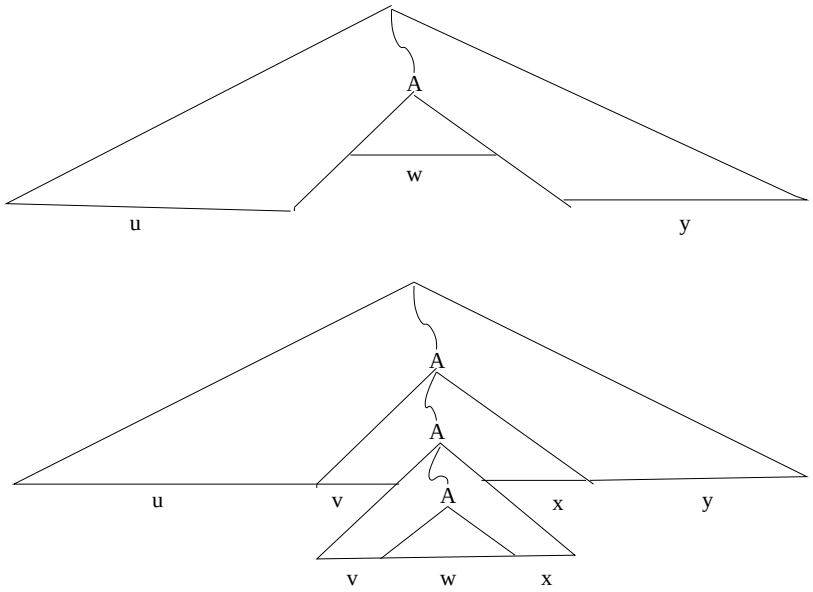
Věta 14.0.9 (Pumping lemma pro bezkontextové jazyky, neboli $uvwxy$-teorém.) *Nechť L je bezkontextový jazyk. Pak existují přirozená čísla p, q tž. každé slovo $z \in L$, $|z| > p$, lze psát ve tvaru $z = uvwxy$, přičemž platí $|vx| \geq 1$ (aspoň jedno ze slov v, x je neprázdné), $|vwx| \leq q$, a pro vš. $i \geq 0$ je $uv^i wx^i y \in L$.*

Důkaz. Čtenář jistě pochopil, že jako ono p můžeme vzít jakékoli číslo větší než délka nejdelšího slova, pro nějž existuje derivační strom bez opakování.

A odkud se vezme ono q zaručující, že lze dokonce omezit délku úseku vwx ? Podstrom na obr. 14 (s kořenem v 'horním' A) lze zvolit tak, že neobsahuje žádné jiné opakování neterminálů – a takových (pod)stromů je zřejmě jen konečně mnoho možných.

Uvedeme nyní podrobnější verzi důkazu s konkrétnějšími odhady čísel p, q.

Nechť $L = L(G)$ pro bezkontextovou gramatiku $G = (\Pi, \Sigma, S, P)$ v CHNF (náležení či nenáležení prázdného slova do L zde nehraje roli).



Předpokládejme, že pro nějaké slovo $z \in L$ existuje derivační strom, v němž se na jedné větvi (tj. cestě od kořene k listu) vyskytují alespoň dva vrcholy označené stejným neterminálem, řekněme A. Pak je zřejmé, že $S \Rightarrow^* uAy \Rightarrow^* uvAxy \Rightarrow^* uvwxy = z$ pro nějaké $u, v, w, x, y \in \Sigma^*$.

Nechť $|\Pi| = k$ (k tedy označuje počet neterminálů). Všimněme si:

a/ na každé větvi derivačního stromu délky alespoň $k + 1$ jsou nejméně dva vrcholy označeny stejným neterminálem;

b/ máme-li derivační strom pro $z \in \Sigma^*$, v němž jsou všechny větve kratší než $k + 1$, pak nutně $|z| \leq 2^{k-1}$;

c/ vezmeme-li lib. $z \in L$ tž. $|z| > 2^{k-1}$ a derivační strom pro z, pak určitě na nejdelší větvi se vyskytují dva různé vrcholy v_1, v_2 (v_1 blíž ke kořeni) označené stejným neterminálem; přitom lze jistě v_1 zvolit tak, že jeho max. vzdálenost k listu je nejvýše $k + 1$. To znamená, že podstrom s kořenem v_1 má nejvýše 2^k listů.

Stačí tedy jako hledaná p, q vzít čísla 2^{k-1} a 2^k.
Konec Důkazu

Pro naše další účely je vhodnější poněkud jednodušší verze věty 14.0.9 (n lze vzít jako $max(p, q) + 1$):

Věta (jiná verze **14.0.9**). *Nechť L je bezkontextový jazyk. Pak existuje přirozené číslo n tž. každé slovo $z \in L$, $|z| \geq n$, lze psát ve tvaru $z = uvwxy$, přičemž platí $|vx| \geq 1$, $|vwx| \leq n$ a pro vš. $i \geq 0$ je $uv^iwx^iy \in L$.*

Všimněte si opět střídání kvantifikátorů:

$(\exists n)\ (\forall z \text{ tž. } z \in L, |z| \geq n)\ (\exists\, u, v, w, x, y \text{ tž. } z = uvwxy, |vwx| \leq n, |vx| \geq 1)\ (\forall i \geq 0) :$

$$uv^iwx^iy \in L$$

A opět se nabízí hra dvou hráčů:

1. A zvolí $n \in \mathcal{N}$
2. B zvolí slovo z tž. $z \in L$ a $|z| \geq n$
3. A zvolí u, v, w, x, y tž. $z = uvwxy$, $|vwx| \leq n$ a $|vx| \geq 1$
4. B zvolí $i \geq 0$

5. *Výsledek:* je-li $uv^iwx^iy \in L$, pak vyhrál A, v případě $uv^iwx^iy \notin L$ vyhrál B.

Je zřejmé, že je-li L bezkontextový, pak A má vítěznou strategii v uvedené hře. Jinak řečeno:

Má-li B vítěznou strategii, pak L není bezkontextový.

Navržení vítězné strategie hráče B je častým prostředkem k důkazu toho, že uvažovaný jazyk není bezkontextový.

Pro výše zkoumaný $L = \{a^nb^nc^n \mid n \geq 0\}$ můžeme vítěznou strategii hráče B formulovat takto.

1. A zvolí (libovolné) $n \in \mathcal{N}$
2. B zvolí $z = a^nb^nc^n$
3. A zvolí libovolné u, v, w, x, y tž. $z = uvwxy$, $|vwx| \leq n$ a $|vx| \geq 1$,
4. B zvolí $i = 0$ (lze také kterékoli $i \geq 2$)
5. Jelikož $|vwx| \leq n$, slova v, x neobsahují aspoň jeden ze symbolů a, b, c (a samozřejmě aspoň jeden obsahují). Proto ve slově uwy nemůže být stejný počet symbolů a, b i c a slovo tedy nepatří do L. B vyhrává.

Poznámka. Z úvodní analýzy (před Větou 14.0.9) víme, že B má vítěznou strategii i při ignorování podmínky $|vwx| \leq n$. Pak sice nemůžeme v posledním bodě jednoduše argumentovat, že v, x neobsahují aspoň jeden ze symbolů a, b, c, ale nepříslušnost slova uwy k L lze dokázat mírně složitějšími úvahami (které jste už, doufejme, provedli před větou 14.0.9). V následujícím příkladu je už podmínka $|vwx| \leq n$ skutečně nutná.

Ukážeme, že jazyk

$$L = \{ww \mid w \in \{0,1\}^*\}$$

není bezkontextový:

1. A zvolí (libovolné) $n \in \mathcal{N}$
2. B zvolí $z = 0^n1^n0^n1^n$
3. A zvolí libovolné u, v, w, x, y tž. $z = uvwxy$, $|vwx| \leq n$ a $|vx| \geq 1$,
4. B zvolí $i = 0$ (lze také kterékoli $i \geq 2$)

5. Jelikož $|vwx| \leq n$, slova v, x zasahují nejvýš do jednoho úseku nul a nejvýš jednoho úseku jedniček v $z = 0^n1^n0^n1^n$ (přičemž alespoň do jednoho úseku zasahují). Tedy $uwy = 0^{k_1}1^{k_2}0^{\ell_1}1^{\ell_2}$, kde určitě $k_1 \neq \ell_1$ nebo $k_2 \neq \ell_2$. Tedy uwy nepatří do L a B vyhrává.

Úkol 64 *Dokažte, že následující jazyky nejsou bezkontextové:*

$L_1 = \{\, 0^m1^n0^m \mid 0 \leq n \leq m \,\}$

$L_2 = \{\, a^k \mid k = n^2 \text{ pro nějaké } n \geq 1 \,\}$

Úkol 65 *Zjistěte, které z daných jazyků*

jsou regulární:

jsou bezkontextové, ale ne regulární:

nejsou bezkontextové:

$L_1 = \{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$ $L_2 = \{w \in \{a,b\}^* \mid |w|_a$ *je sudé* $\}$
$L_3 = \{w \in \{a,b\}^* \mid w$ *obsahuje podslovo abba*$\}$ $L_4 = \{w \in \{a,b,c\}^* \mid |w|_a = |w|_b = |w|_c\}$ $L_5 = \{w \in \{a,b\}^* \mid |w|_a$ *je prvočíslo*$\}$ $L_6 = \{\, 0^m1^n \mid m \leq 2n \,\}$
$L_7 = \{\, 0^m1^n0^m \mid m = 2n \,\}$

Kapitola 15

Uzávěrové vlastnosti třídy bezkontextových jazyků

Zkratkou CFL budeme označovat třídu bezkontextových jazyků. CFL není uzavřena na všechny operace, na které je uzavřena třída regulárních jazyků. Nejdříve si ukážeme případy operací, vůči nimž CFL uzavřena je. Důkazy jsou samozřejmě opět konstruktivní – ukazují algoritmy, které k zadané reprezentaci jazyků-operandů zkonstruují reprezentaci jazyka-výsledku operace. (Příslušnou reprezentací jsou samozřejmě bezkontextové gramatiky či zásobníkové automaty ; lze volit, co je vhodnější.)

Věta 15.0.10 *CFL je uzavřena vůči sjednocení, zřetězení, iteraci, zrcadlovému obrazu, substituci (tedy i homomorfismu).*

Důkaz. K bezkontextovým gramatikám $G_1 = (\Pi_1, \Sigma, S_1, P_1)$, $G_2 = (\Pi_1, \Sigma, S_1, P_1)$ lze zkonstruovat gramatiku $G = (\Pi, \Sigma, S, P)$ tž. $L(G) = L(G_1) \cup L(G_2)$ takto: Předpokládáme, že $\Pi_1 \cap \Pi_2 = \emptyset$ (docílíme toho případným přejmenováním neterminálů). Položíme $\Pi = \Pi_1 \cup \Pi_2 \cup \{S\}$, kde $S \notin \Pi_1 \cup \Pi_2$, a $P = P_1 \cup P_2 \cup \{S \longrightarrow S_1, S \longrightarrow S_2\}$.

Rovněž velmi přímočará je konstrukce gramatik generujících jazyky $L(G_1) \cdot L(G_2)$, $L(G_1)^*$, $L(G_1)^R$.

Podobně ke gramatice $G = (\Pi, \Sigma, S, P)$ a gramatikám G_a (pro vš. $a \in \Sigma$) lze snadno zkonstruovat gramatiku, která generuje jazyk vzniklý z $L(G)$ substituujeme-li za každé a jazyk $L(G_a)$.
Konec Důkazu

V důkazu další uzávěrové věty se více hodí reprezentace jazyků automaty.

Věta 15.0.11 *CFL je uzavřena vůči průniku s regulárním jazykem, i vůči kvocientu podle regulárního jazyka. (Tj. pro každý bezkontextový L a regulární R, jsou $L \cap R$, $R\backslash L$, L/R bezkontextové.)*

Důkaz. Idea pro průnik:
Lze podobně jako u dvou KA, zde lze příslušný KA "zabudovat" do řídicí jednotky ZA. (Stavová množina výsledného ZA je kartézským součinem stavových množin původního ZA a KA.)

Idea pro kvocient je:
Mějme ZA M a KA A. Připomeňme, že slovo u patří do $L(A)\backslash L(M)$ právě když ex. $v \in L(A)$ tak, že $vu \in L(M)$. Pro vytvoření ZA M' přijímající jazyk $L(A)\backslash L(M)$ opět použijeme myšlenku zabudování řídicí jednotky A do řídicí jednotky M. Nyní ale tak, že výsledný ZA M' dělá na začátku sérii ε-kroků, při nichž nedeterministicky "hádá" vhodné v.
(Zkuste dokončit promyšlení detailů konstrukce.)
Konec Důkazu

Neuzavřenost CFL vůči některým operacím se nejpříměji dokáže konstrukcí (jednoduchých) protipříkladů; je samozřejmě možné užít i další úvahy:

Věta 15.0.12 *CFL není uzavřena vůči průniku a doplňku.*

Důkaz. Jazyky $L_1 = \{a^i b^j c^k \mid i = j\}$, $L_2 = \{a^i b^j c^k \mid j = k\}$ jsou zřejmě bezkontextové. Přitom $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$ bezkontextový není.

Z de Morganových pravidel plyne, že kdyby byla CFL uzavřena vůči doplňku, tak by díky uzavřenosti vůči sjednocení byla uzavřena i vůči průniku.
Konec Důkazu

Kapitola 16

Deterministické zásobníkové automaty

Přednáška 12

Připomeňme si, že u zásobníkových automatů jsme jako základní vzali *nedeterministickou* verzi. Takto totiž ZA odpovídají bezkontextovým gramatikám. Vzniká přirozená otázka, jak to vypadá s deterministickou verzí zásobníkových automatů – determinismus je navíc potřebný, máme-li na takovém zařízení opravdu stavět (rychlý) *algoritmus* (např. již zmíněné syntaktické analýzy). Začneme s definicí *deterministického* zásobníkového automatu a deterministického bezkontextového jazyka:

Definice 16.1 *Deterministický zásobníkový automat (DZA) je ZA* $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$*, pro nějž platí:*

1. $\delta(q, a, X)$ *je vždy nejvýše jednoprvková množina (pro* $a \in \Sigma \cup \{\varepsilon\}$*) a*
2. *je-li* $\delta(q, \varepsilon, X) \neq \emptyset$*, pak* $\delta(q, a, X) = \emptyset$ *pro vš.* $a \in \Sigma$*.*

Jazyk L *je* deterministický bezkontextový jazyk*, jestliže* $L = L_{KS}(M)$ *pro nějaký DZA* M*.*

Smysl je jasný: pro každé vstupní slovo existuje jediný možný výpočet DZA M. Všimněme si, že v případě přijímání prázdným zásobníkem je přijímaný jazyk $L_{PZ}(M)$ nutně *bezprefixový* – pro slovo $u \in L_{PZ}(M)$ každý jeho vlastní prefix nepatří do $L_{PZ}(M)$. (Např. jazyk $\{\varepsilon, a\}$ bezprefixový není.) Proto jsou deterministické jazyky, tvořící třídu DCFL, definovány pomocí přijímání koncovým stavem.

Využitím "bottom-symbolu" lze opět snadno ukázat, že ke každému DZA M lze zkonstruovat DZA M' tž. $L_{PZ}(M) = L_{KS}(M')$. Na druhé straně lze ke každému DZA M zkonstruovat DZA M' tž. $L_{KS}(M) \cdot \{\$\} = L_{PZ}(M')$, kde \$ je přidaný koncový znak.

Na rozdíl od konečných automatů, deterministická verze zásobníkových automatů je skutečně slabší než nedeterministická, tedy DCFL je *vlastní* podtřídou CFL. Lze to vidět už díky jiným uzávěrovém vlastnostem třídy DCFL.

Věta 16.0.13 *Třída DCFL je uzavřena vůči doplňku. Na druhé straně není uzavřena vůči průniku ani vůči sjednocení.*

Uzavřenost vůči doplňku nelze sice demonstrovat prostým prohozením přijímajících a nepřijímajících stavů (proč ?), není ale těžké tuto myšlenku "dotáhnout". Neuzavřenost vůči průniku plyne např. z toho, že jazyky L_1, L_2 z důkazu věty 15.0.12 jsou deterministické. DCFL tedy nemůže být uzavřena ani vůči sjednocení (de Morganova pravidla).

Uzávěrové vlastnosti lze např. využít pro důkazy nepříslušnosti některých jazyků k DCFL. Např. jazyk $L = \{a^i b^j c^k \mid (i \neq j) \vee (j \neq k)\}$ není v DCFL (přitom zřejmě je v CFL): Kdyby byl, pak by i jeho doplněk $\overline{L}$ byl v DCFL, tedy i v CFL. Pak by ovšem i $\overline{L} \cap [a^*b^*c^*]$ byl v CFL (CFL je uzavřena vůči průniku s regulárním jazykem); ovšem $\overline{L} \cap [a^*b^*c^*] = \{\, a^n b^n c^n \mid n \geq 0\}$, a tedy bezkontextový není !

Využitím dalších uzávěrových vlastností se dá ukázat, že např. jazyky $\{ww^R \mid w \in \{a, b\}^*\}$, $\{a^i b^j c^k \mid (i = j) \vee (j = k)\}$ nejsou deterministické.

Vzpomeňme si, že existuje algoritmus, který o dvou zadaných konečných automatech rozhodne, zda jsou ekvivalentní (tj. zda přijímají tentýž jazyk). V kursu o vyčíslitelnosti a složitosti uvidíme, že podobný algoritmus pro (nedeterministické) zásobníkové automaty neexistuje. Od 60. let ale byla otevřena otázka, zda existuje algoritmus rozhodující ekvivalenci deterministických zásobníkových automatů . Pozitivní řešení prezentoval v r. 1997 G. Sénizergues, později důkaz zjednodušil C. Stirling. (Uvedení důkazu v našem kursu však pro jeho náročnost stále nepřipadá v úvahu.)

Kapitola 17

Chomského hierarchie

Dříve uvedené bezkontextové gramatiky jsou speciálním případem obecných (generativních) gramatik. Od bezkontextových se liší jen tím, že na levé straně pravidel nestojí nutně jen jeden neterminál, ale obecně řetězec neterminálů a terminálů obsahující alespoň jeden neterminál.

Pro úplnost uvádíme úplnou obecnou definici:

Definice 17.1 *Generativní gramatika je dána čtveřicí parametrů* $G = (\Pi, \Sigma, S, P)$*, kde* Π *je konečná množina* neterminálních symbolů *(*neterminálů*),* Σ *je konečná množina* terminálních symbolů *(*terminálů*), přičemž* $\Pi \cap \Sigma = \emptyset$*,* $S \in \Pi$ *je* počáteční (startovací) neterminál *a* P *je konečná množina pravidel typu* $\alpha \to \beta$*, kde* $\alpha \in (\Pi \cup \Sigma)^* \Pi (\Pi \cup \Sigma)^*$ *a* $\beta \in (\Pi \cup \Sigma)^*$*.*

Uvažujme lib. $\gamma, \delta \in (\Pi \cup \Sigma)^*$*. Řekneme, že* γ se přímo přepíše *(*lze přímo přepsat*) na* δ *(podle pravidel gramatiky* G*), značíme* $\gamma \Rightarrow_G \delta$ *nebo jen* $\gamma \Rightarrow \delta$ *(když* G *zřejmá z kontextu), jestliže existují slova* $\mu_1, \mu_2, \alpha, \beta$ *tž.* $\gamma = \mu_1 \alpha \mu_2$*,* $\delta = \mu_1 \beta \mu_2$ *a* $\alpha \to \beta$ *je pravidlo v* P*.*

Řekneme, že γ se přepíše *na* δ*, značíme* $\gamma \Rightarrow^* \delta$*, jestliže existuje posloupnost* $\mu_0, \mu_1, \ldots, \mu_n$ *slov z* $(\Pi \cup \Sigma)^*$ *(pro něj.* $n \geq 0$*) tž.* $\gamma = \mu_0 \Rightarrow \mu_1 \Rightarrow \ldots \Rightarrow \mu_n = \delta$*. Zmíněnou posloupnost pak nazveme* odvozením *(*derivací*) délky* n *slova* δ *ze slova* γ*.*

Jazyk generovaný gramatikou G*, označme jej* $L(G)$*, je definován takto:* $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$*.*

Dvě gramatiky G_1, G_2 *nazveme* ekvivalentní*, jestliže* $L(G_1) = L(G_2)$*.*

Tzv. Chomského hierarchie vzniká omezením se na speciální typ pravidel:

Definice 17.2 *Obecná generativní gramatika* $G = (\Pi, \Sigma, S, P)$ *definovaná výše je* gramatika typu 0 *v tzv.* Chomského hierarchii*.* G *je* gramatika typu

1*, neboli kontextová gramatika, jestliže každé pravidlo v* P *je tvaru* $\alpha X \beta \to \alpha \gamma \beta$*, kde* $|\gamma| \geq 1$ *(připomeňme, že* $\alpha, \beta, \gamma \in (\Pi \cup \Sigma)^*, X \in \Pi$*); jedinou výjimkou může být pravidlo* $S \to \varepsilon$*, v kterémžto případě se pak* S *nesmí vyskytovat na pravé straně žádného pravidla.* G *je* gramatika typu 2*, neboli bezkontextová gramatika, jestliže každé pravidlo v* P *je tvaru* $X \to \alpha$*.* G *je* gramatika typu 3*, neboli regulární gramatika, jestliže každé pravidlo v* P *je tvaru* $X \to wY$ *nebo* $X \to w$ *(*$w \in \Sigma^*$*).*

Jazyk L *je* typu i *(*$i = 0, 1, 2, 3$*) v Chomského hierarchii, jestliže jej generuje nějaká gramatika typu* i*. Speciálně řekneme, že* jazyk *je* kontextový *(*bezkontextový, regulární*), jestliže jej generuje nějaká kontextová (bezkontextová, regulární) gramatika.*

Všimněme si, že gramatika typu 3 je speciálním případem gramatiky typu 2, gramatika typu 1 je spec. případem gramatiky typu 0. Gramatika typu 2 (bezkontextová gramatika) nemusí být gramatikou typu 1 kvůli pravidlům s ε na pravé straně; je ji však možné do takové formy upravit (připomeňme si konstrukci nevypouštějící bezkontextové gramatiky). Označíme-li tedy $\mathcal{L}_i$ třídu jazyků typu i ($i = 0, 1, 2, 3$), pak je zřejmé

Lemma 17.0.14 $\mathcal{L}_3 \subseteq \mathcal{L}_2 \subseteq \mathcal{L}_1 \subseteq \mathcal{L}_0$

Ve skutečnosti jsou všechny inkluze vlastní, jak ještě zmíníme později.

Kapitola 18

Konečné automaty a regulární gramatiky

Teď si ukážeme, že nová definice regulárního jazyka nekoliduje s dřívější definicí; začneme technickým lemmatem.

Lemma 18.0.15 *Ke každé regulární gramatice, lze zkonstruovat ekvivalentní gramatiku, jejíž každé pravidlo je v jednom z tvarů $X \to aY$, $X \to Y$, $X \to \varepsilon$.*

Důkaz. Pravidlo typu $X \to a_1 a_2 \dots a_n Y$ $(n \geq 2)$ nahradíme pravidly $X \to a_1 Z_1$, $Z_1 \to a_2 Z_2$, …, $Z_{n-1} \to a_n Y$, kde $Z_1, Z_2, \dots, Z_n$ jsou vždy nově přidané neterminály.
Konec Důkazu

Věta 18.0.16 *Jazyk je generován regulární gramatikou právě když je rozpoznáván konečným automatem.*

Důkaz. Nechť $A = (Q, \Sigma, \delta, q_0, F)$ je KA. Sestrojme $G = (Q, \Sigma, q_0, P)$, kde do P zařadíme $q \to aq'$ pro každé q, q', a tž. $\delta(q, a) = q'$ a navíc přidáme $q \to \varepsilon$ pro každé $q \in F$. Je snadné ověřit, že G je

regulární gramatika tž. $L(G) = L(A)$; indukcí je např. možné dokázat vztah $(\delta(q, w) = q') \Leftrightarrow (q \Rightarrow_G^* wq')$.

Naopak uvažujme gramatiku $G = (\Pi, \Sigma, S, P)$ s pravidly typu $X \to aY$, $X \to Y$, $X \to \varepsilon$. Sestrojme ZNKA $A = (\Pi, \Sigma, \delta, \{S\}, F)$, kde $Y \in \delta(X, a)$ $(a \in \Sigma \cup \{\varepsilon\})$ právě když $X \to aY$ patří do P. Navíc $F = \{X \mid (X \to \varepsilon) \in P\}$. Opět je snadné ověřit $L(A) = L(G)$.

Konec Důkazu

Úkol 66 *Rozšiřte konstrukci převodu KA na RG pro případ nedeterministického KA a aplikujte ji v případě NKA zadaného tabulkou.*

	a	*b*
↔*1*	-	*4*
→*2*	*2,3*	*1*
3	*3*	*1*
←*4*	*3*	*3,4*

Úkol 67 *K uvedené regulární gramatice sestrojte ekvivalentní nedeterministický konečný automat.*

$S \longrightarrow abS \,|\, bbaA \,|\, \varepsilon$
$A \longrightarrow abA \,|\, bB$
$B \longrightarrow acS \,|\, bC \,|\, \varepsilon$
$C \longrightarrow aC \,|\, bA$

Kapitola 19

Turingovy stroje

Víme už, že jazyky třídy $\mathcal{L}_3$ jsou charakterizovány konečnými automaty, jazyky třídy $\mathcal{L}_2$
(nedeterministickými) zásobníkovými automaty. Nyní uvedeme výpočetní model, který se ukáže být ekvivalentní obecným gramatikám, a sice tzv. Turingovy stroje.

Turingův stroj je podobný konečnému automatu, rozdíl je v tom že páska, na níž je na začátku zapsáno vstupní slovo (ostatní buňky jsou prázdné, tj. je v nich zapsán speciální prázdný znak), je oboustranně nekonečná, hlava spojená s konečnou řídicí jednotkou se může pohybovat po pásce oběma směry a je nejen čtecí, ale i *zapisovací* – symboly v buňkách pásky je tedy možné přepisovat, a to i jinými než vstupními symboly. Formalizujme nyní pojem Turingova stroje, jeho výpočtu a jazyka jím přijímaného.

Definice 19.1 *Turingův stroj, zkráceně TS, M je určen následujícími parametry (přesněji řečeno, je to uspořádaná šestice) $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, kde*

- Q *je konečná neprázdná množina* stavů
- Γ *je konečná neprázdná množina* (páskových) symbolů
- $\Sigma \subseteq \Gamma$, $\Sigma \neq \emptyset$ *je množina* vstupních symbolů *(tzv. vstupní abeceda)*
- $q_0 \in Q$ *je* počáteční stav,
- $F \subseteq Q$ *je množina* koncových stavů
- $\delta : (Q - F) \times \Gamma \to Q \times \Gamma \times \{-1, 0, +1\}$ *je přechodová funkce*

Předpokládáme, že v $\Gamma - \Sigma$ je vždy obsažen speciální prvek # označující prázdný znak.

Konfigurací Turingova stroje M *rozumíme libovolné slovo tvaru uqv, kde $u, v \in \Gamma^*$ a $q \in Q$.* Konfigurace uqv *je* počáteční, *jestliže $u = \varepsilon$, $q = q_0$ a $v \in \Sigma^*$; uqv je* koncová konfigurace, *jestliže $q \in F$.*

Konfigurace uqv a $\#^i uqv\#^j$ $(i, j \geq 0)$ ztotožňujeme, speciálně tedy konfiguraci qv ztotožňujeme s konfigurací $\#qv$, podobně uq s $uq\#$.

Konfigurace $K = uaqbv$, *kde $u, v \in \Gamma^*$, $a, b \in \Gamma$* vede v jednom kroku ke konfiguraci K', *příslušnou relaci označujeme $\vdash_M$ nebo jen $\vdash$ (píšeme tedy $K \vdash K'$) právě když platí jedna z těchto možností:*

- $\delta(q, b) = (q', b', 0)$ *a* $K' = uaq'b'v$,
- $\delta(q, b) = (q', b', +1)$ *a* $K' = uab'q'v$,
- $\delta(q, b) = (q', b', -1)$ *a* $K' = uq'ab'v$.

Relace $\vdash^*$ je reflexivním a tranzitivním uzávěrem relace $\vdash$.

Slovo $u \in \Sigma^*$ je přijímáno *TS M, jestliže $q_0 u \vdash^* K$ pro nějakou koncovou konfiguraci K.*

Jazykem přijímaným *TS M rozumíme jazyk $L(M) = \{w \in \Sigma^* \mid w$ je přijímáno $M\}$.*

Všimněte si, že výpočet pro danou počáteční konfiguraci je jediný (stroj je deterministický), nemusí však skončit ! Podle naší definice slovo není přijímáno strojem M právě tehdy, když je výpočet nad ním nekonečný (na rozdíl od KA či ZA, koncový stav je u TS 'opravdu' koncový, výpočet dále nemůže pokračovat; pro nekoncový stav je další krok vždy definován).

Čtenáře asi napadne varianta definice, která by vyžadovala, aby každý výpočet TS vždy skončil, a sice buď ve speciálním *přijímajícím stavu* (vstupní slovo přijato) nebo *zamítajícím stavu* (vstupní slovo zamítnuto). Jazyky, které je možné Tur. stroji takto *rozhodovat* (nazývají se také *rekurzivní* nebo *rozhodnutelné jazyky*) jsou ovšem vlastní podtřídou jazyků přijímaných Tur. stroji (které jsou také nazývány *rekurzivně spočetné* či *částečně rozhodnutelné jazyky*). Důkaz bude proveden v kursu o vyčíslitelnosti a složitosti. (Tam se mj. také ukáže, že neexistuje algoritmus, který by pro zadaný Turingův stroj zjistil, zda (každý) jeho výpočet skončí.)

Přednáška 13

Turingovy stroje patří mezi tzv. *univerzální výpočetní modely*, tj. ty, které jsou schopny realizovat jakýkoli algoritmus (to je obsahem tzv. Church-Turingovy teze, o níž bude podrobněji pojednáno v kursu o vyčíslitelnosti a složitosti). Mj. to znamená, že obohacení uvedeného modelu např. o další pásky, další (čtecí a zapisovací) hlavy, nebo přidání programových konstrukcí jako např. *if ... then*, *while ... do* apod. vede sice k jednoduššímu zápisu algoritmů, ale nikoli k rozšíření třídy přijímaných jazyků (či obecněji třídy vyčíslitelných [realizovatelných] funkcí); standardní model Turingova stroje dokáže všechny tyto rozšířené modely simulovat. Podrobněji bude o této problematice pojednáno v kursu o vyčísltelnosti a složitosti, teď si jen stručně všimneme rozšíření vzniklého využitím nedeterminismu.

Úkol. Využitím zkušeností s konečnými a zásobníkovými automaty nadefinujte pojem *nedeterministických Turingových strojů* a jazyků jimi přijímaných.

Věta 19.0.17 *Třída jazyků přijímaných (determimistickými) Turingovými stroji se rovná třídě jazyků přijímaných nedetermimistickými Turingovými stroji.*

Důkaz. Idea:

Pro daný nedeterministický TS M lze snadno sestavit algoritmus, který pro zadané vstupní slovo w systematicky zkoumá všechny výpočty TS M délky 1, pak všechny výpočty délky 2, pak všechny výpočty délky 3 atd. (jinak řečeno: strom možných výpočtů M nad w je prohledáván 'do šířky'). Pro zadané w uvedený algoritmus nutně objeví přijímající výpočet stroje M nad w, jestliže takový existuje; v takovém případě algoritmus skončí (a slovo w přijme), jinak běží donekonečna. Algoritmus pak stačí 'naprogramovat' jako deterministický Turingův stroj.
Konec Důkazu

Kapitola 20

Další poznámky ke vztahu automatů a gramatik

Pojem nedeterministického Turingova stroje lze např. využít pro očividný důkaz jednoho směru následující věty:

Věta 20.0.18 *Jazyky přijímané Turingovými stroji jsou právě jazyky typu 0.*

Úkol 68 *Který směr je ten očividný ? Vysvětlete proč.*

Idea důkazu druhého směru: relace $\vdash_M$ je de facto relací přepisování mezi slovy jisté konečné abecedy, určené konečně mnoha pravidly. K M lze tedy sestavit obecnou gramatiku, která je schopna, zhruba řečeno, vygenerovat slovo wXw (pro lib. w), v 'pravé kopii' pak odsimulovat výpočet stroje M nad w a v případě, že tento skončí, smaže se symbol X se vším napravo.

Pro úplnost dodejme, že kontextové jazyky (jazyky typu 1) jsou charakterizovány tzv. lineárně omezenými automaty, LBA (linear bounded automata), tj Turingovými stroji, které používají jen úsek pásky, v němž je zapsáno vstupní slovo. (Je možné si představit, že vstupní slovo je ohraničeno spec. symboly, levou a pravou zarážkou; ty nemohou být přepsány a z levé (pravé) zarážky je možný jen pohyb doprava (doleva). 'Základní' verze automatu je ovšem *nedeterministická*; problém, zda DLBA (deterministické LBA) přijímají tytéž jazyky jako LBA je dlouhodobě otevřený.

Věta 20.0.19 *Jazyk je kontextový (tj. typu 1) právě tehdy, když je přijímán nějakým LBA.*

Jeden směr je opět přímočarý (zjistíte, který ?), druhý je více techničtější.

Zmínili jsme již, že inkluze v Tvrzení 17.0.14 jsou vlastní. Víme např., že jazyk $\{a^n b^n \mid n \geq 0\}$ je typu 2 ale nikoli 3, a také, že $\{a^n b^n c^n \mid n \geq 0\}$ není typu 2 – je ovšem zřejmé, že je typu 1. Existenci jazyka v $\mathcal{L}_0 - \mathcal{L}_1$ lze ukázat např. diagonalizační metodou, o níž pojednáme v kursu o vyčíslitelnosti a složitosti.

Všimněme si ještě, že u TS si lze pásku vlevo od hlavy a pásku vpravo od hlavy představit jako dva zásobníky. Vyvoďte z toho, že model "zásobníkový automat s dvěma zásobníky" (nadefinujte jej formálně !) přijímá tytéž jazyky jako Turingovy stroje.
