OSTRAVSKÁ UNIVERZITA V OSTRAVĚ

TEORETICKÉ ZÁKLADY INFORMATIKY II

HASHIM HABIBALLA

OSTRAVA 2003

Tento projekt byl spolufinancován Evropskou unií a českým státním rozpočtem

Recenzenti:

Doc. Ing. Miroslav Beneš, Ph.D.
PhDr. Danuše Matýsková

Název:	Teoretické základy informatiky II
Autoři:	Mgr. Hashim Habiballa
Vydání:	první, 2003
Počet stran:	61
Náklad:	50
Tisk:	Ediční středisko CIT OU

Studijní materiály pro distanční kurz: Teoretické základy informatiky II

Jazyková korektura nebyla provedena, za jazykovou stránku odpovídá autor.
Určeno výhradně pro kurzy Celoživotního vzdělávání Moravskoslezska

Vydavatel a tisk: Ostravská univerzita v Ostravě,
Systém celoživotního vzdělávání Moravskoslezska

ISBN 80-7042-861-9

Teoretické základy informatiky II.

Hashim Habiballa

Cíl předmětu

Navázat na kurz Teoretické základy informatiky I. – pokračovat vyšší třídou jazyků – bezkontextovými jazyky. Příklady aplikace teoretické informatiky ve výuce algoritmizace. Software pro podporu výuky.

Formální jazyky a gramatiky. Bezkontextové gramatiky a jazyky. Chomského hierarchie. Aplikace v praxi, použití programátorských pomůcek ve výuce teoretické informatice.

Obsah:

1. Bezkontextové gramatiky a jazyky, regulární gramatiky 5
   1.1. Bezkontextová gramatika a bezkontextový jazyk 6
   1.2. Tvorba gramatik k bezkontextovým jazykům 8
   1.3. Regulární gramatiky, vztah k regulárním jazykům 10
2. Nevypouštějící a redukované gramatiky 16
   2.1. Nevypouštějící gramatiky 16
   2.2. Převody na nevypouštějící a redukované tvary 18
3. Kanonická odvození, jednoznačné gramatiky 21
   3.1. Kanonické derivace a nejednoznačnost 21
4. Normální formy, lemma o vkládání 23
   4.1. Chomského normální forma a pumping lemma 23
   4.2. Greibachové normální forma 26
5. Zásobníkové automaty 32
   5.1. Zásobníkový automat a vztah k BKJ 33
6. Vlastnosti tříd bezkontextových jazyků 39
   6.1. Uzávěrové vlastnosti třídy BKJ 39
7. Chomského hierarchie 41
   7.1. Obecná generativní gramatika a Chomského hierarchie 41
   7.2. Turingův stroj 43
8. Aplikace v programátorských úlohách 45
   8.1. Regulární jazyky a konečné automaty v praxi 45
   8.2. Bezkontextové gramatiky a syntaktická analýza 48
9. Programátorské didaktické pomůcky 51
   9.1. Počítačové aplikace jako programátorské didaktické pomůcky 51
   9.2. PregJaut 52
   9.3. GramAut 54
   9.4. RABJ 55
10. Vybrané partie teorie vyčíslitelnosti a složitosti 57
   10.1. Vyčíslitelnost 57
   10.2. Složitost 57

Úvod pro práci s textem pro distanční studium.

Průvodce studiem:

Účel textu

Tento text navazuje na studijní oporu Teoretické základy informatiky II. Pokračuje ve výkladu teorie formálních jazyků a automatů – především bezkontextovými gramatikami a jazyky. Dále obsahuje některé úlohy, které lze realizovat ve výuce algoritmizace, na nichž mohou studenti nejen lépe pochopit pokročilé programátorské techniky, ale zároveň se tak seznámí nenásilnou formou s některými pojmy teorie formálních jazyků, jako jsou regulární výrazy a konečné automaty.

Struktura

V textu jsou dodržena následující pravidla:

- je specifikován cíl lekce (tedy co by měl student po jejím absolvování umět, znát, pochopit)
- výklad učiva
- řešené příklady a úlohy k zamyšlení
- důležité pojmy – otázky
- korespondenční úkoly (mohou být sdruženy po více lekcích)

Pokuste se tuto oporu využívat nejen při studiu dalších teoretických témat, ale využijte ji pro prohloubení učiva z prvního dílu opory. K tomu Vám poslouží především software, který máte k dispozici v balíčku studijních materiálů a pomůcek. Na konci textu najdete návod, jak tyto pomůcky používat.

Zpracujte prosím všechny korespondenční úkoly a zašlete mi je včas – každá lekce by měla představovat zhruba jeden týden v prezenčním studiu a z toho vycházejte při jejich odesílání. Prosím respektujte, že distanční forma studia je náročná nejen pro studenty, ale i pro mne jako tutora.

Pokud si text budete prohlížet jako dokument formátu PDF, aktivujte si v nastavení vyhlazování vektorové (čárové) grafiky, jinak budete mít některé obrázky nepříjemně „zubaté“

Zároveň Vás chci laskavě poprosit, abyste jakékoliv věcné i formální chyby v textu zdokumentovali a kontaktovali mne.Budu Vám za tuto pomoc vděčen a usnadní to učení i Vašim kolegům.

1. Bezkontextové gramatiky a jazyky, regulární gramatiky

Cíl:

Po prostudování této kapitoly pochopíte:
- co je bezkontextová gramatika
- jak pojem gramatiky souvisí s jazykem
- vztah regulárních gramatik k regulárním jazykům

Naučíte se:

- tvořit automaty pro jednoduché bezkontextové jazyky
- převádět regulární gramatiky na automaty a naopak

Nyní se v našem studiu dostáváme do druhé části. V předchozím díle jsme se zabývali třídou jazyků rozpoznatelných konečnými automaty resp. regulárními jazyky. Viděli jste, že existují i jazyky, které nejsou regulární. Konečné automaty jsou pro ně příliš „slabé“, nedokáží je rozpoznávat a regulární výrazy je nemohou generovat. Proto by bylo rozumné se ptát, zda neexistují nástroje, které nám umožní tyto jazyky generovat a analyzovat. Takové nástroje existují a postupně se s nimi i s jejich vlastnostmi seznámíme a opět se naučíte vytvářet k jazykům jejich instance.

Složitější jazyky než regulární

Těmito nástroji jsou bezkontextová gramatika a zásobníkový automat. Pojem gramatiky jsme si již částečně objasnili na intuitivním příkladě v kapitole 2 prvního dílu. Je to prostředek, jak na základě pravidel lze generovat (terminální) slova v jisté (terminální) abecedě pomocí postupného dosazování do neterminálních symbolů (proměnných). Občerstvěte si tyto pojmy v paměti. Zásobníkový automat je konečný automat, který má však navíc možnost pracovat s jistým druhem paměťového zařízení – zásobníkem, na který si může ukládat symboly a vybírat je zpět. Nicméně může tak činit pouze přístupem – poslední dovnitř, první ven (to znamená může zapisovat jen na vrchol zásobníku – struktura LIFO, jak ji znáte z algoritmizace). Naučíte se tyto struktury používat a také si ukážeme jejich speciální tvary. Stejně jako u regulárních jazyků si pak ukážeme, že jejich výpočetní síla – třída jazyků rozpoznatelných zásobníkovými automaty a bezkontextových jazyků generovaných bezkontextovými gramatikami – je totožná.

Podívejme se nejprve na příklad, jak se s bezkontextovými gramatikami pracuje a pak si zavedeme jejich formální definice. Uváděli jsme si, že jazyk $L=\{0^n1^n;\ n \geq 0\}$ není rozpoznatelný konečným automatem. Existuje však velice jednoduchá bezkontextová gramatika, která tento jazyk generuje:

Tato gramatika má vstupní abecedu (terminálních symbolů) – {0,1}, abecedu proměnných (neterminálů) – {S}, neterminál, od kterého se generování (odvozování) vždy začíná určený jako S a tato dvě pravidla, určující možnosti „dosazení“ za proměnné:
$S \rightarrow 0S1$, $S \rightarrow \varepsilon$.
Tato pravidla umožňují buď dosadit za S řetězec 0S1 (obsahuje opět v sobě S), nebo ukončit toto generování slovem prázdným. Díky tomu, že vždy ke každé nule na levé straně slova dodáme jedničku na pravé straně slova, můžeme si takto „napumpovat“ kolik chceme nul a jedniček, ovšem jedině ve stejném počtu. Toto konečný automat ani regulární výraz neuměl. V gramatice pak můžeme provádět odvození terminálních slov (která budou všechna vytvářet jazyk), např. takto
$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 0011$
(v posledním kroku jsme použili pravidlo na prázdné slovo, čímž jsme se zbavili S a dostali slovo složené jen z terminálů).

1.1. Bezkontextová gramatika a bezkontextový jazyk

Jak uvidíme, pojem gramatiky lze zobecnit. Tím dosáhneme i daleko větší síly než poskytuje bezkontextová gramatika. Obecný pojem gramatiky:
Gramatika G je určena konečnou množinou *neterminálů* (neterminálních symbolů - proměnných), konečnou množinou *terminálů*, která nemá společné prvky s množinou neterminálů, *počátečním neterminálem* a konečnou *soustavou* (množinou) *přepisovacích pravidel* typu $\alpha \rightarrow \beta$, (α přepiš na β), kde α,β jsou řetězece z neterminálů a terminálů, navíc $\alpha \neq \varepsilon$.
Gramatika, označme ji G, může být chápána jako čtveřice $G=(\Pi,\Sigma,S,P)$
Π je množina neterminálů,
Σ je množina terminálů, $\Pi \cap \Sigma = \varnothing$
$S \in \Pi$ je počáteční neterminál,
P je konečná množina přepisovacích pravidel.

Bezkontextová gramatika se od tohoto obecného pojmu odlišuje tím, že připouští, aby přepisovací pravidlo mělo pouze tvar $X \rightarrow \alpha$, to znamená že pouze můžeme přepisovat vždy jednu proměnnou na řetězec proměnných i terminálů.

Bezkontextová gramatika

Definice 1: ***Bezkontextová gramatika*** **(BKG) je určena konečnou množinou *neterminálů* (neterminálních symbolů - proměnných), konečnou množinou *terminálů*, která nemá společné prvky s množinou neterminálů, *počátečním neterminálem* a konečnou *soustavou* (množinou) *přepisovacích pravidel* typu $X \rightarrow \alpha$, (X přepiš na α), kde X je neterminál a α je řetězec z neterminálů a terminálů.**
Bezkontextová gramatika, označme ji G, může být chápána jako čtveřice $G=(\Pi,\Sigma,S,P)$

Π je množina neterminálů,
Σ je množina terminálů, $\Pi \cap \Sigma = \varnothing$
$S \in \Pi$ je počáteční neterminál,
P je konečná množina přepisovacích pravidel.

V takto definované gramatice můžeme pak odvozovat slova, jak jsme viděli na příkladu.

Odvození v gramatice

Definice 2: Nechť $G=(\Pi,\Sigma,S,P)$ je bezkontextová gramatika a nechť $\alpha,\beta \in (\Pi \cup \Sigma)^*$.

1. **Řekneme, že α *se přímo přepíše na* β podle pravidel gramatiky G, označíme $\alpha \Rightarrow_G \beta$ (nebo $\alpha \Rightarrow \beta$, pokud je zřejmé o jakou G se jedná), právě tehdy, když existuje $\gamma_1,\gamma_2,\delta \in (\Pi \cup \Sigma)^*$ a $X \in \Pi$ takové, že:**

$\alpha = \gamma_1 X \gamma_2$; $\beta = \gamma_1 \delta \gamma_2$; $X \rightarrow \delta$ patří do P

2. **Řekneme, že α *se přepíše na* β, značíme $\alpha \Rightarrow_G^* \beta$ (nebo $\alpha \Rightarrow^* \beta$), jestliže existuje posloupnost (*) $\gamma_0,\gamma_1,\gamma_2 \ldots \gamma_n$ prvků $(\Pi \cup \Sigma)^*$ (pro nějaké $n \geq 0$) taková, že:**

$\alpha = \gamma_0 \Rightarrow_G \gamma_1 \Rightarrow_G \gamma_2 \Rightarrow_G \gamma_3 \Rightarrow_G \ldots \Rightarrow_G \gamma_{n-1} \Rightarrow_G \gamma_n = \beta$

3. **Posloupnost (*) nazveme *odvození* (*derivace*) slova β ze slova α.**
4. ***Jazyk generovaný gramatikou* G, označme jej L(G), je definován následovně:**

$L(G)=\{ w | w \in \Sigma^* \text{ a } S \Rightarrow_G^* w \}$

Stejně jako u konečných automatů, můžeme definovat pojem ekvivalentních gramatik. Opět půjde o gramatiku, která generuje stejný jazyk. Příkladem budiž ekvivalentní gramatika ke gramatice v příkladu:
$S \rightarrow ASB$, $S \rightarrow \varepsilon$, $A \rightarrow 0$, $B \rightarrow 1$ (přidali jsme neterminály A,B)

Definice 3: Bezkontextové gramatiky G_1,G_2 nazveme *ekvivalentní*, právě když $L(G_1)=L(G_2)$.

Definice 4: *Bezkontextový jazyk (BKJ)* je jazyk (jazyk $L \subseteq \Sigma^*$ pro nějakou konečnou abecedu Σ) generovaný nějakou bezkontextovou gramatikou (tedy L=L(G) pro nějakou bezkontextovou gramatiku G).

Ekvivalentní gramatika, Bezkontextový jazyk

Bezkontextový jazyk tedy tvoří všechna terminální slova, která můžeme odvodit z počátečního neterminálu.

Poznámka:

1. $\Rightarrow^*$ je reflefivní a tranzitivní uzávěr relace $\Rightarrow$.
2. $\alpha \Rightarrow_G^* \beta$ často čteme "α generuje β", "z α se odvodí β" apod.
3. odvození (*) nazveme minimální, jestliže $\gamma_i \neq \gamma_j \ \forall i \neq j$; je zřejmé, že když $\alpha \Rightarrow^* \beta$, pak lze β odvodit z α nějakým minimálním

odvozením. Dále budeme odvozením (derivací) myslet minimální odvození.

4. obvykle

Konvence

jednotlivé terminály značíme - a,b,c...

řetězce terminálů značíme - u,v,w...

jednotlivé neterminály - A,B,C... X,Y,Z

řetězce neterminálů a terminálů - α,β,γ...

Prázdné slovo budeme někdy označovat také jako e, stejně jako tomu bylo již u automatů.

Poznámka: Bezkontextovou gramatiku obvykle budeme zadávat pouze množinou přepisovacích pravidel. Budeme-li neterminály označovat velkými písmeny, terminály jinak a počáteční neterminál S, pak budou všechny parametry gramatiky zřejmé.

1.2. Tvorba gramatik k bezkontextovým jazykům

Podívejme se na některé řešené příklady tvorby bezkontextových gramatik:

Řešený příklad 1:
Sestrojte bezkontextovou gramatiku G_i tak, aby $L(G_i)=L_i$; $1 \le i \le 7$

$L_1=\{w \in \{0,1\}^*; w=1^k0^k; k \ge 0\}$

Řešení: G_1:
$S \rightarrow 1S0$, $S \rightarrow e$.

Tento příklad nám ukazuje, jak lze realizovat konstrukci pro typický příklad jazyka, který není regulární, jak jsme poznali v předcházejících kapitolách.Chcete-li zaručit stejný počet symbolů na opačných stranách slova, pak je musíte v jednom pravidle uvést na příslušných místech od stejného neterminálu (to samozřejmě není jediný způsob – ale ostatní budou principiálně podobné).

Řešený příklad 2:
$L_2=\{w \in \{0,1\}^*; w=(011)^j101^k0^{k+1}; j,k \ge 0\}$

Řešení: G_2:
$S \rightarrow ABC$, $A \rightarrow 011A$, $A \rightarrow e$, $B \rightarrow 10$,$C \rightarrow 1C0$, $C \rightarrow 0$.

Vidíte, že stejně jako u automatů je někdy dobré si složitý problém rozdělit na podproblémy. Zde jsme si rozdělili generování celého slova na neterminály ABC. A generuje regulární jazyk A = [(011)*], regulární jazyk (jedno slovo) B = [10], bezkontextový jazyk, který již umíme generovat (téměř) C = $\{ 1^k0^{k+1}, k \geq 0\}$. Poslední zmiňovaný jazyk je jednoduchou modifikací z ilustrativního příkladu z počátku kapitoly. Liší se tím, že generování nekončí prázdným slovem, ale symbolem 0, který má být na pravé straně vždy navíc.

Řešený příklad 3:
$L_3=\{w \in \{0,1\}^*; w=1uu^R0;u \in \{0,1\}^+\}$

Řešení: G_3:
$S\rightarrow 1A0$, $A\rightarrow 1B1$, $A\rightarrow 0B0$, $B\rightarrow e$,$B\rightarrow 1B1$, $B\rightarrow 0B0$.

Pokud chcete zajistit jako v tomto případě, aby se rozpoznával zrcadlový obraz (tedy čím u začíná tím u^R končí, atd...), pak stačí v pravidlech vždy ke stejnému symbolu na počátku vygenerujeme stejný na konci. Takto se nám napumpují na bocích zrcadlově stejná slova.

Řešený příklad 4:
$L_4=\{w \in \{a,b\}^*; w=aba^* \text{ nebo } w=(ab)^*bab\}$

Řešení: G_4:
$S\rightarrow A$, $S\rightarrow B$, $A\rightarrow ab$, $A\rightarrow Aa$,$B\rightarrow bab$, $B\rightarrow abB$.

Opět jde o případ dekompozice problému. Máme zde dvě podmínky a stačí, aby byla splněna jedna z nich. Jinak řečeno, vygeneruje buď slovo podle 1. nebo 2. podmínky. Proto už od začátku můžeme tyto dvě alternativní cesty v gramatice zohlednit – tedy S se přepisuje buď na A nebo B.

Kontrolní úkoly:

Sestrojte gramatiky pro následující jazyky:

$L_5=\{w \in \{a,b,c\}^*; w=a^ib^{i+2}uabcu^R; i \geq 1; u \in \{a,b\}^*\}$

$L_6=\{w \in \{0,1\}^*; w=(011)^i(110)^j(11)^{2j}; i,j \geq 0\}$

$L_7=\{w \in \{a,b\}^*; w=(ab)^k(bab)^j; j \geq 0; k \geq j\}$

Řešení:

G_5: $S\rightarrow AB$, $A\rightarrow abbb$, $A\rightarrow aAb$, $B\rightarrow abc$,$B\rightarrow aBa$, $B\rightarrow bBb$.
G_6: $S\rightarrow AB$, $A\rightarrow 011A$, $A\rightarrow e$, $B\rightarrow e$,$B\rightarrow 110B1111$.
G_7: $S\rightarrow abSbab$, $S\rightarrow abS$, $S\rightarrow e$.

1.3. Regulární gramatiky, vztah k regulárním jazykům

Bezkontextové jazyky lze dále omezit až na přesně třídu regulárních jazyků (jinak řečeno za jistých podmínek BKG generují jazyky rozpoznatelné KA). Stačí omezíme-li pravidla BKG na taková, která přepisují neterminál na slovo terminálů a za ním následuje maximálně jeden neterminál. Taková formalizace odpovídá tomu, co rozpoznává KA. Uvidíte, jak se dá velmi jednoduše ke KA sestrojit regulární gramatika a naopak. Vychází se z toho, že můžeme stavy konečného automatu považovat za neterminály, symboly u přechodů za začátek přepisovaného slova a stav, do kterého se jde, za neterminál za terminálním slovem. Je-li stav výstupní, generování slova může skončit a proto se tento stav (neterminál) přepíše na ε.

Regulární gramatika

Definice 5: Bezkontextová gramatika G=(Π,Σ,S,P) se nazývá *regulární gramatika*, jestliže každé pravidlo v P je v jednom z tvarů $X \rightarrow wY$, $X \rightarrow w$, kde $X,Y \in \Pi$, $w \in \Sigma^*$.

***Jazyk* L nazveme *regulární*, jestliže je generován nějakou regulární gramatikou (tedy L=L(G) pro nějakou regulární gramatiku G).**

Ukážeme, že definice nekoliduje s dřívější definicí regulárního jazyka.

Jazyky generované regulárními gramatikami

Věta 1: Každý jazyk rozpoznatelný konečným automatem je regulární (ve smyslu definice pomocí regulární gramatiky).

Důkaz:

1.

Mějme libovolný jazyk, označme jej L. Předpokládejme, že jazyk L je rozpoznáván nějakým konečným automatem, označme jej A. Ukážeme jak k automatu A zkonstruovat regulární gramatiku, označme ji G, která generuje jazyk L, tím bude důkaz hotov.

Mějme tedy nějak zadán automat A. Je tedy nějakým způsobem určena (vymezena) množina jeho stavů, dále je určena množina vstupních symbolů, jeden stav je označen za počáteční, některé stavy za koncové. Přitom je zadán předpis, který pro každý stav a každý vstupní symbol udává, do kterého stavu automat A přejde přečtením onoho vstupního symbolu nachází-li se v onom stavu.

Způsob převodu KA na BKG

Pro názornost uvedeme příklad, kde A je zadán tabulkou:

	$Q \backslash^{\Sigma}$	0	1
↔	1	1	2
	2	1	3
	3	1	3

Budeme konstruovat gramatiku G.

Nejprve označme všechny stavy automatu A např. písmeny $A_1,A_2,... A_n$, kde n je počet stavů.
V našem příkladu tedy:

	$Q\backslash^{\Sigma}$	0	1
↔	A_1	A_1	A_2
	A_2	A_1	A_3
	A_3	A_1	A_3

Symboly $A_1,A_2,... A_n$ budou tvořit množinu neterminálů gramatiky G. Symbol označující počáteční stav, bude počátečním neterminálem gramatiky G, v našem příkladu tedy A_1. Množina terminálů gramatiky G bude totožná se vstupní abecedou automatu A ({ 0,1} v našem příkladu).
Množinu přepisovacích pravidel konstruujeme následovně.
Vezmeme symbol A_1. Dále vezmeme některý terminál, označme jej a a zjistíme stav, do kterého automat A přejde ze stavu A_1 přečtením terminálu a. Tento stav nechť je označen A_j. Pak mezi přepisovací pravidla zahrneme pravidlo $A_1 \rightarrow aA_j$.

V příkladu pro a=0 dostaneme pravidlo $A_1 \rightarrow 0A_1$. Totéž provedeme pro všechny ostatní terminály (a symbol A_1). V příkladu tedy ještě přidáme pravidlo $A_1 \rightarrow 1A_2$. Pak celý postup opakujeme pro A_2, pak pro A_3 atd., až pro A_n.

V příkladu tedy takto vytvoříme pravidla:
$A_1 \rightarrow 0A_1$, $A_1 \rightarrow 1A_2$, $A_2 \rightarrow 0A_1$, $A_2 \rightarrow 1A_3$, $A_3 \rightarrow 0A_1$, $A_3 \rightarrow 1A_3$.
Nakonec přidáme pravidla, která umožňují smazat, neboli přepsat na prázdné slovo, všechny ty neterminály, které označují koncové stavy automatu A. V příkladu tedy přidáme jen jedno pravidlo, a to $A_1 \rightarrow e$.
Tím jsme vytváření přepisovacích pravidel, a tím i celé gramatiky G ukončili.
Všimněme si, že každému "výpočtu" automatu A znázorněnému
$A_{i0} \rightarrow^{a}{}_1 A_{i1} \rightarrow^{a}{}_2 A_{i2} \rightarrow^{a}{}_3 ... \rightarrow^{a}{}_m A_{im}$
odpovídá v gramatice G odvození
$A_{i0} \Rightarrow a_1 A_{i1} \Rightarrow a_1 a_2 A_{i2} \Rightarrow ... \Rightarrow a_1 a_2 a_3 ... a_m A_{im}$
V příkladu např.
$A_1 \rightarrow^{0} A_1 \rightarrow^{1} A_2 \rightarrow^{1} A_3$

$A_1 \Rightarrow 0 A_1 \Rightarrow 01 A_2 \Rightarrow 011 A_3$
Když si nyní uvědomíme, že počáteční neterminál gramatiky G odpovídá počátečnímu stavu automatu A, a dále, že smazat lze jedině neterminály odpovídající koncovým stavům, je zřejmé, že každý přijímající výpočet automatu A (pro nějaké slovo, označme ho w) odpovídá odvození slova w z počátečního neterminálu v gramatice G a naopak. Tím jsme se přesvědčili, že gramatika G skutečně generuje jazyk L (rozpoznávaný automatem A).
2.(formálně zapsáno)

Nechť L=L(A) pro konečný automat A=(Q,Σ,δ,q_0,F). Sestrojme gramatiku G=(Q,Σ,q_0,P), kde P={ q→aq′|δ(q,a)=q′}∪{ q→ e |q ∈ F}.
Snadno lze ověřit, že pro libovolné w ∈ Σ^* platí w ∈ L(A)⇔($q_0 \Rightarrow_G^*$ wq pro něj. q ∈ F)⇔($q_0 \Rightarrow_G^*$ w)⇔ w ∈ L(G).

Stejně jako lze k automatu sestrojit gramatiku, lze i ke gramatice sestrojit automat. Postup je v podstatě reverzí postupu, který jste se právě naučili. Jelikož však v obecné regulární gramatice jsou celá slova, která bychom nemohli do přechodů přímo zapsat, je třeba formulovat tvrzení, že každá taková gramatika se může převést na gramatiku s pravidly, kde je nejvýše jeden terminální symbol před neterminálem.

Věta 2: Ke každé regulární gramatice existuje ekvivalentní regulární gramatika, která má pravidla pouze následující typů:
X→ aY, X→ Y, X→ e
kde X,Y jsou neterminály a a je terminál.

Důkaz:
Nechť G=(Π,Σ,S,P) je regulární gramatika.
Sestrojíme G′=(Π′,Σ,S,P′) následovně:

1. Do P′ zahrneme všechna pravidla z P, která jsou v jednom z povolených typů:

X→ aY, X→ Y, X→ e

2. Místo každého pravidla z P tvaru X→ a_1 a_2...a_mY (m ≥ 2) zahrneme do P′ soustavu pravidel X→a_1 Y_1, Y_1→ a_2 Y_2, Y_2→ a_3 Y_3 ... Y_{m-2}→ $a_{m-1}Y_{m-1}$, Y_{m-1}→ a_m Y, kde Y_1,Y_2,Y_3... Y_{m-1} jsou nově přidané neterminály.
3. Místo každého pravidla typu X→ a_1 a_2 ... a_n (n ≥ 2) zahrneme do P′ soustavu pravidel X→a_1 Y_1, Y_1→ a_2 Y_2, Y_2→ a_3 Y_3 ... Y_{n-1}→ a_n Y_n, Y_n→ e, kde Y_1,Y_2,Y_3... Y_n jsou nově přidané neterminály.

Π′ obsahuje neterminály z Π a všechny nově přidané neterminály.
Je zřejmé, že G′ je požadovaného typu a také lze snadno ověřit, že L(G)=L(G′).

Věta 3: Každý regulární jazyk (ve smyslu definice podle regulární gramatiky) je rozpoznatelný konečným automatem.

Regulární gramatika -> KA

Důkaz:
Nechť L=L(G) pro regulární gramatiku G=(Π,Σ,S,P). Podle předchozí věty, lze předpokládat, že pravidla G jsou pouze typů

X→ aY, X→ Y, X→ e
Sestrojme zobecněný nedeterministický konečný automat A=(Q,Σ,δ,I,F), kde Q=Π; I={S}, F={ q|(q→ e) ∈ P} a ∀q ∈ Q(=Π), ∀a ∈ Σ je δ(q,a)={ q′|(q → aq′) ∈ P} a δ(q,e)={ q′|(q → q′) ∈ P}.
Ověření L(G)=L(A) je podobné jako u důkazu věty 54.

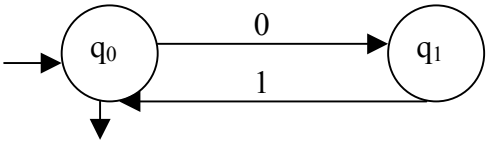
Tento reverzibilní postup nyní demonstrujeme na tomto kontrolním úkolu:

Kontrolní úkol:

Mějme regulární gramatiku:
$S \rightarrow 01S$, $S \rightarrow \varepsilon$, která rozpoznává jazyk $L = \{(01)^n , n \geq 0\}$.

Abychom mohli tuto gramatiku převést na automat, musíme nejprve postupem dle důkazu věty ji převést (na tvar $X \rightarrow aY$, $X \rightarrow Y$, $X \rightarrow e$).
$S \rightarrow 0A$, $A \rightarrow 1S$, $S \rightarrow \varepsilon$,

Automat pak vypadá takto:



Z tohoto automatu pak můžeme zpětně zase dojít k této gramatice.

Viděli jste, že BKG může mít jisté omezení pravidel, které ovlivňuje jazyky, které taková gramatiky generují. Například jazyk $L = \{ 0^n1^n , n \geq 0\}$ logicky nemůžeme generovat regulární gramatikou, neboť není regulární. Dalším tvarem pravidel je levá lineární gramatika. Její odvozovací síla je opět stejná jako u regulární, neboť pravidla si lze představit jako zrcadlové obrazy, ke kterým je lehké sestrojit automat rozpoznávající zrcadlový obraz.

Definice 6: ***Levá lineární gramatika*** **je bezkontextová gramatika, jejíž pravidla jsou pouze typu: $X \rightarrow Yw$ nebo $X \rightarrow w$, kde X,Y jsou neterminály a w je řetězec terminálů.**

Věta 4: **Jazyk je regulární, právě když je generován nějakou levou lineární gramatikou.**

Levá lineární gramatika

Důkaz:
Nechť $G=(\Pi,\Sigma,S,P)$ je levá lineární gramatika. Sestrojme gramatiku $G'=(\Pi,\Sigma,S,P')$ tak, že platí $X \rightarrow \alpha \in P \Leftrightarrow X \rightarrow \alpha^R \in P'$ $(\forall X \in \Pi, \alpha \in (\Pi \cup \Sigma^*))$
Je zřejmé, že $L(G)=(L(G'))^R$, přitom G' je regulární (pravá lineární). Věta tedy plyne z uzavřenosti třídy regulárních jazyků vůči zrcadlovému obrazu.

Dalším omezením je lineární gramatika, která již nemá stejnou sílu jajko regulární. Například právě pro $L = \{ 0^n1^n , n \geq 0\}$ lze sestrojit gramatiku lineární, ale nikoliv regulární.

Lineární gramatika

Definice 7: ***Lineární gramatika*** **je bezkontextová gramatika, jejíž pravidla jsou pouze typu X→ uYv nebo X→ u, kde X,Y jsou neterminály a u,v jsou řetězce terminálů.**

Jazyk je *lineární*, je-li generován nějakou lineární gramatikou.

Věta 5: **Třída regulárních jazyků je vlastní podtřídou třídy lineárních jazyků.**

Důkaz:
Že je podtřídou je zřejmé z definice, že je vlastní podtřídou, ukazuje např. jazyk $\{ 0^n 1^n | n \geq 0\}$ (S→ 0S1|e).

Poznámka: Ukážeme, že se obejdeme bez X→ e - vypouštějícího pravidla.

Řešený příklad 5:
Sestrojte regulární gramatiku G tak, aby
a)L(G)=[$a^*b+(bab)^*bb$]

Řešení:
G:
S→ A, S→ B, A→ aA, A→ b, B→ babB, B→ bb.

b)L(G)=[$((00)^*+(11)^*)101(110)^*$]

Řešení:
G:
S→ A, S→ B, A→ 00A, A→ C, B→ 11B, B→ C, C→ 101D, D→ 110D, D→ e.

c)L(G)=[$abab^*+((bb)^*a)^*$]

Řešení:
G:
S→ A, S→ B, A→ abaC, B→ e, B→ D, C→ bC, C→ e, D→ bbD, D→ aE, E→ e, E→ D.

Řešený příklad 6:
Určete jazyk, který generuje bezkontextová gramatika G.
a) G:S→ 11S0, S→ e.

Řešení:
$L(G)=\{w \in \{0,1\}^*; w=(11)^j0^j; j \geq 0\}=$
$=\{w \in \{0,1\}^*; w=1^{2j}0^j; j \geq 0\}$

b) G:S→ ABC, A→ bbA, A→ ccB,B→ bBb, B→ a, C→ aCa, C→ bb.

Řešení:
$L(G)=\{w \in \{a,b,c\}^*; w=(bb)^i ccb^j ab^j b^k ab^k a^m bba^m; i,j,k,m \geq 0\}$

c) $G:S \rightarrow 001,\ S \rightarrow 01S,\ S \rightarrow 01S1.$

Řešení:
$L(G)=\{w \in \{0,1\}^*; w=(01)^j 001(1)^i; i \geq 0; j \geq i\}$

Nejdůležitější probrané pojmy:

- bezkontextová gramatika, bezkontextový jazyk
- odvození
- regulární gramatika, lineární gramatika

Úkoly k textu:

1. Sestrojte DKA rozpoznávající jazyk $L=\{w \in \{a,b\}^* | w$ končí symbolem ‚a' nebo obsahuje ‚bab'$\}$ a převeďte ho na regulární gramatiku.
2. Lze každý ZNKA převést na regulární gramatiku? Zdůvodněte proč.

2. Nevypouštějící a redukované gramatiky

Cíl:

Po prostudování této kapitoly pochopíte:
- pojem nevypouštějící BKG
- pojem redukované gramatiky

Naučíte se:

- převádět BKG na nevypouštějící gramatiky
- převádět BKG na redukované gramatiky

Bezkontextové gramatiky mohou mít své speciální tvary. Tyto speciální tvary nemusejí porušovat třídu jazyků, které rozpoznávají obecné BKG a mohou mít některé výhodné vlastnosti. Například prázdné slovo v pravidlech může být někdy na obtíž a působit komplikace při převodech. Ukážeme si, že lze formulovat tvar bez e-pravidel a každou gramatiku lze do tohoto tvaru převést. Princip spočívá v tom, že odhalíme ty neterminály, které se přepisují na e a ty pak v ostatních pravidlech vynecháme (všechny kombinace i s původním pravidlem).

2.1. Nevypouštějící gramatiky

Definice 8: Bezkontextová gramatika se nazývá *nevypouštějící*, jestliže neobsahuje žádné pravidlo typu $X \rightarrow e$, kde X je neterminál.

Nevypouštějící gramatika

Věta 6: Ke každé bezkontextové gramatice G lze zkonstruovat nevypouštějící gramatiku G′ takovou, že $L(G')=L(G)-\{e\}$.

Důkaz:
Mějme $G=(\Pi,\Sigma,S,P)$. Zkonstruujme množinu U; $U=\{X \in \Pi | X \Rightarrow_G^* e\}$. U lze zkonstruuovat např. následovně. Položme $U_1=\{X \in \Pi | (X \rightarrow e) \in P\}$, obecně $U_{i+1}=U_i \cup \{X \in \Pi | X \rightarrow \alpha$, kde $\alpha \in U_i^*$, je pravidlo v $P\}$ $(i \geq 1)$.
Zřejmě je $U_1 \subseteq U_2 \subseteq U_3 \subseteq ... \subseteq \Pi$. Pro nějaké n je $U_n=U_{n+1}$ a podle konstrukce je pak zřejmé, že $U_n=U_{n+k}$ pro libovolné $k \geq 0$. Tedy $U=U_n$.
Zkonstruujme gramatiku $G_1=(\Pi,\Sigma,S,P_1)$ následovně: v P_1 jsou právě taková pravidla $X \rightarrow \alpha$, pro která $\alpha \neq e$, přičemž pro každé takové $X \rightarrow \alpha$ existuje v P pravidlo $X \rightarrow \beta$, kde α vznikne z β vynecháním některých (třeba žádných) výskytů neterminálů z množiny U.
Chceme ukázat, že $L(G_1)=L(G)-\{e\}$.
Ukažme nejdříve $L(G_1) \subseteq L(G)$. Jestliže G_1 vygeneruje w, pak w je generováno i gramatikou G. Nová pravidla můžeme simulovat starými,

přičemž používáme generování prázdného slova. Jelikož e $\notin$ $L(G_1)$, pak je zřejmé, že $L(G_1) \subseteq L(G)-\{e\}$.
Nyní předpokládáme, že w $\neq$ e je odvozeno v gramatice G. Fakt, že w lze odvodit i v G_1 je zřejmý z toho, že výskyty neterminálů, které se v odvození podle G přepíší na prázdné slovo, může odvození v G_1 rovnou vynechat.

Věta 7: **Ke každé bezkontextové gramatice G=(Π,Σ,S,P) existuje ekvivalentní gramatika G′=(Π∪{S′},Σ,S′,P′) taková, že e se může vyskytovat na pravé straně jedině v pravidle S′→ e, přičemž S′ se nevyskytuje na pravé straně žádného pravidla P′.**

Důkaz:
Jestliže e $\notin$ L(G), pak je tvrzení zřejmé z předchozí věty. Jestliže e $\in$ L(G), postupujeme následovně:
Vezměme $G_1=(\Pi,\Sigma,S,P_1)$ tak jako v důkazu předchozí věty. Nyní stačí položit $G'=(\Pi\cup\{S'\},\Sigma,S',P_1\cup\{S'\rightarrow S, S'\rightarrow e\})$.

Dalším speciálním tvarem je redukovaná gramatika. Stejně jako u automatů, mohou i v gramatice existovat zbytečné neterminály (stavy). Jde v zásadě o dva případy. Buď některý neterminál nemůže vygenerovat žádné slovo například se cyklí sám v sobě nebo se na některý neterminál nedá vůbec dostat s počátečního S. Opět lze každou gramatiku na tento tvar převést. V první fázi hledáme ty „neproduktivní“ neterminály (vyjdeme od těch, které se přímo přepisují na terminální slovo) a v druhé ty „nedosažitelné“ (princip je podobný hledání nedosažitelných stavů automatu).

Definice 9: **Bezkontextová gramatika G=(Π,Σ,S,P) se nazývá *redukovaná*, jestliže platí následující dvě podmínky:**
1. Pro každé X $\in$ Π existuje w $\in$ Σ^* takové, že $X \Rightarrow_G^* w$.
2. Pro každé X $\in$ Π existují $\alpha,\beta \in (\Pi\cup\Sigma)^*$ takové, že $S \Rightarrow_G^* \alpha X \beta$.

Redukovaná gramatika

Věta 8: **Ke každé bezkontextové gramatice G takové, že L(G) ≠ ∅, lze zkonstruovat ekvivalentní redukovanou gramatiku.**

Důkaz:
Nechť G=(Π,Σ,S,P).
1. Zkonstruujeme množinu U; $U=\{X \in \Pi | X \Rightarrow_G^* w$ pro nějaké w $\in \Sigma^*\}$. U lze zkonstruovat např. takto: položme $U_0=\Sigma$, $U_{i+1}=U_i \cup \{X \in \Pi | (X \rightarrow \alpha) \in P$ pro nějaké $\alpha \in U_i^*\}$, i ≥ 0; $U_0 \subseteq U_1 \subseteq U_2 \subseteq U_3 \subseteq ... \subseteq \Pi\cup\Sigma$. Když $U_n=U_{n+1}$ pak $\forall k \geq 0$ $U_n=U_{n+k}$. Stačí vzít $U=U_n - \Sigma$.
Z pravidel gramatiky G vyhodíme všechna pravidla, obsahující nějaký neterminál z Π− U. Tím obdržíme gramatiku G′ takovou, že L(G)=L(G′). Navíc G′ splňuje vlastnost 1. z definice redukované gramatiky.

2. Zkonstruujeme množinu V; $V=\{X \in U | \exists \alpha,\beta \in (U \cup \Sigma)^*$ tak, že $S \Rightarrow_G^* \alpha X \beta\}$. Z pravidel gramatiky G′ odstraníme všechna pravidla obsahující nějaký neterminál z U – V. Tím dostaneme gramatiku G″ takovou, že $L(G)=L(G'')$. Navíc G″ splňuje podmínku 2. z definice redukované gramatiky a přitom se neporušila platnost bodu 1.
(G″ splňuje 1. i 2. a je tedy redukovaná.)

Z následujících tvrzení vyplývá, že lze zjistit zda gramatika generuje prázdný jazyk. Pokud ji převedeme na redukovanou, zjistíme jednoduše, zda obsahuje nějaký neterminál nebo ne.

Věta 66.: Existuje algoritmus, který pro libovolnou bezkontextovou gramatiku G rozhodne, zda $L(G)=\emptyset$.

Rozhodnutelnost $L(G)=\emptyset$

Důkaz:
Důsledek části 1. předchozího důkazu.

2.2. Převody na nevypouštějící a redukované tvary

Řešený příklad 7:

Sestrojte bezkontextovou gramatiku G_1 tak, aby $L(G_1)=L(G)-\{e\}$ a aby G_1 byla nevypouštějící.
a)G:
$S \rightarrow ABC$, $A \rightarrow 011A$, $A \rightarrow e$, $B \rightarrow 10$, $C \rightarrow 1C0$, $C \rightarrow 0$.

Řešení: $U_1=\{A\}$, $U_2=\{A\}$, $U=\{A\}$.
Pak G_1 bude vypadat takto:
$S \rightarrow ABC$, $S \rightarrow BC$, $A \rightarrow 011A$, $A \rightarrow 011$, $B \rightarrow 10$, $C \rightarrow 1C0$, $C \rightarrow 0$.

b)G:
$S \rightarrow AB$, $A \rightarrow 011A$, $A \rightarrow e$, $B \rightarrow e$, $B \rightarrow 110B1111$.

Řešení: $U_1=\{A,B\}$, $U_2=\{A,B,S\}$, $U_3=\{A,B,S\}$, $U=\{A,B,S\}$.
Pak G_1 bude vypadat takto:
$S \rightarrow AB$, $S \rightarrow B$, $S \rightarrow A$, $A \rightarrow 011A$, $A \rightarrow 011$, $B \rightarrow 110B1111$, $B \rightarrow 1101111$.

c)G:
$S \rightarrow AB$, $A \rightarrow abbb$, $A \rightarrow aAb$, $B \rightarrow abc$, $B \rightarrow aBa$, $B \rightarrow bBb$.
Řešení: Gramatika je nevypouštějící. $G_1=G$.

d)G:

S→ A, S→ B, A→ abaC, B→ e, B→ D, C→ bC, C→ e, D→ bbD,D→ aE, E→ e, E→ D.

Řešení: U_1={B,C,E}, U_2={B,C,E,S}, U_3={B,C,E,S}, U={B,C,E,S}.
Pak G_1 bude vypadat takto:
S→ A, S→ B, A→ abaC, A→ aba, B→ D, C→ bC, C→ b, D→ bbD,D→ aE, D→ a, E→ D.

Řešený příklad 8:

Sestrojte ekvivalentní redukovanou bezkontextovou gramatiku G_r k bezkontextové gramatice G.
a) G:S→ ABC, S→ a, A→ aA,A→ aB, B→ bBb, B→ Bb, C→ cAB, C→ c.

Řešení:
U_0={a,b,c}, U_1={a,b,c,S,C}, U_2={a,b,c,S,C}, U={S,C}
G_t:S→ a, C→ c.
V={S}
G_r:S→ a.
(A platí:L(G)=L(G_t)=L(G_r)={a}.)

b) G:S→ XY, S→ YZ, X→ aX,X→ e, Y→ bYb, Y→ X, Z→ bZ, M→ b, M→ bMc.

Řešení:
U_0={a,b,c}, U_1={a,b,c,X,M}, U_2={a,b,c,X,M,Y}, U_3={a,b,c,X,M,Y,S}, U_4=U_3, U={X,M,Y,S}
G_t:S→ XY, X→ aX,X→ e, Y→ bYb, Y→ X, M→ b, M→ bMc.
V={S,X,Y}
G_r:S→ XY, X→ aX,X→ e, Y→ bYb, Y→ X.
(A platí:L(G)=L(G_t)=L(G_r)={$a^m b^j a^k b^j$;m,j,k ≥ 0}.)

Nejdůležitější probrané pojmy:

- nevypouštějící gramatika
- redukovaná gramatika

Korespondenční úkol:

Část 1:
Vyberte si dva redukované automaty z tohoto textu a převeďte je na regulární gramatiky.

Část 2:

Navrhněte dvě gramatiky s nejméně pěti neterminály. Převeďte je na redukovanou a nevypouštějící formu. Gramatiky musí mít taková pravidla, aby se při vytváření redukované gramatiky alespoň jeden neterminál zredukoval při každé fázi.
Určete jaký jazyk tuto gramatiky generují.

3. Kanonická odvození, jednoznačné gramatiky

Cíl:

Po prostudování této kapitoly pochopíte:
- kanonická odvození
- nejednoznačnost gramatik

Naučíte se:
- vytvářet strom odvození

Odvození v lineární textové podobě není jediná možnost, jak ho reprezentovat. Další možnost je vytvořit strom odvození. Odvození však u některých gramatik pro stejné slovo může být různé. Mluvíme pak o nejednoznačných gramatikách.

3.1. Kanonické derivace a nejednoznačnost

Často je výhodné omezit se na tzv. *kanonické derivace*, tj. levé nebo pravé derivace. Nejde o nic jiného než o konvenci, kterou dodržujeme během celého odvození v gramatice. Buď se rozhodneme, že vždy přepíšeme neterminál nejvíce vpravo v odvozovaném slově nebo ten nejvíce vlevo.

Levé a pravé odvození (kanonická)

Definice 10: Odvození v bezkontextové gramatice se nazývá *levé odvození* (levá derivace), jestliže se v něm přepisuje vždy nejlevější neterminál. Odvození v bezkontextové gramatice se nazývá *pravé odvození* (pravá derivace), jestliže se v něm přepisuje vždy nejpravější neterminál.

Podrobněji: Nechť G=(Π,Σ,S,P) je bezkontextová gramatika. Řekneme, že α se *přepíše levým přepsáním* na β ($\alpha,\beta \in (\Pi\cup\Sigma)^*$), jestliže existuje $X\rightarrow\gamma \in P$ takové, že $\alpha = uX\delta$, $\beta = u\gamma\delta$ ($u \in \Sigma^*$, $\delta \in (\Pi\cup\Sigma)^*$).

O odvození $\alpha_0\Rightarrow\alpha_1\Rightarrow\alpha_2\Rightarrow...\Rightarrow\alpha_n$ řekneme, že je to *levá derivace* jestliže α_i se přepíše na α_{i+1} levým přepsáním pro všechna i=0,1,2... n−1.

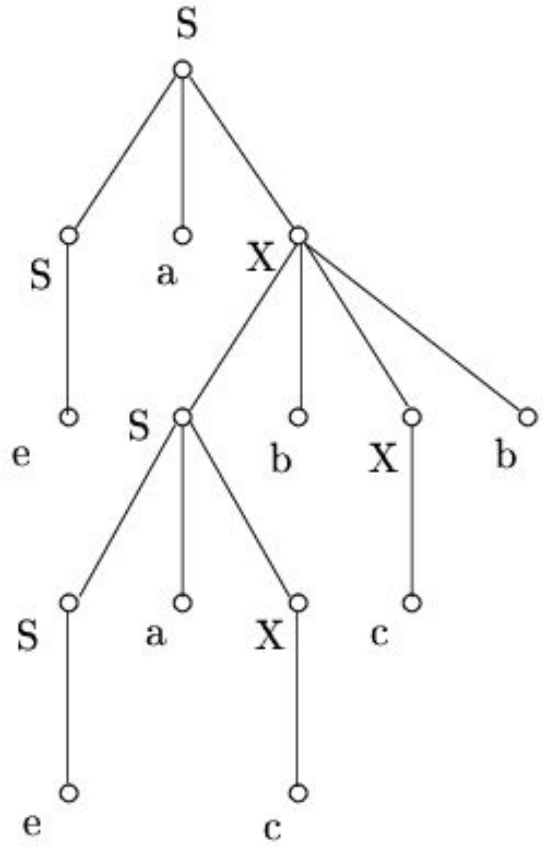
Podobně pravá derivace:

Nechť G=(Π,Σ,S,P) je bezkontextová gramatika. Řekneme, že α se *přepíše pravým přepsáním* na β ($\alpha,\beta \in (\Pi\cup\Sigma)^*$), jestliže existuje $X\rightarrow\gamma \in P$ takové, že $\alpha = \delta Xu$, $\beta = \delta\gamma u$ ($u \in \Sigma^*$, $\delta \in (\Pi\cup\Sigma)^*$).

O odvození $\alpha_0\Rightarrow\alpha_1\Rightarrow\alpha_2\Rightarrow...\Rightarrow\alpha_n$ řekneme, že je to *pravá derivace* jestliže α_i se přepíše na α_{i+1} pravým přepsáním pro všechna i=0,1,2... n−1.

Věta 9: Nechť G=(Π,Σ,S,P) je bezkontextová gramatika. Jestliže $X\Rightarrow_G^* w$ ($X \in \Pi, w \in \Sigma^*$), pak w je z X odvoditelné nějakým levým (pravým) odvozením.

Poznámka: Ke každému odvození slova z jazyka generovaného gramatikou, existuje derivační strom daného odvození. Nebudeme uvádět přesnou definici derivačního stromu - zůstane na intuitivní úrovni.
Derivační strom na obr. je derivačním stromem např. odvození S⇒ SaX⇒ aX⇒ aSbXb⇒aSaXbXb⇒ aaXbXb⇒ aacbXb⇒ aacbcb v gramatice G obsahující určitě alespoň pravidla S→ SaX|e a pravidla X→ SbXb|c, ale tento derivační strom je taktéž derivačním stromem odvození S⇒ SaX⇒ SaSbXb⇒ aSbXb⇒aSaXbXb⇒ aaXbXb⇒ aacbXb⇒ aacbcb



Nejednoznačná gramatika a jazyk

Definice 11: Bezkontextová gramatika je *nejednoznačná*, jestliže pro některé slovo w ∈ L(G) existují dvě různé levé derivace (dva různé derivační stromy). V opačném případě je gramatika *jednoznačná*.

Poznámka: Bezkontextový jazyk, který nelze nagenerovat jednoznačnou gramatikou se nazývá ***vnitřně nejednoznačný*.**

4. Normální formy, lemma o vkládání

Cíl:

Po prostudování této kapitoly pochopíte:
- co je Chomského normální forma
- co je Greibachové normální forma
- co je pumping lemma (lemma o vkládání)

Naučíte se:
- aplikovat pumping lemmu na důkazy, že jazyk není bezkontextový
- převádět BKG na Greibachové normální formu

V této kapitole si ukážeme, jak mohou vypadat některé speciální tvary gramatik - normální formy a jak se na ně převádí. Zároveň se naučíte, jak dokazovat, že jazyky nejsou bezkontextové. Je to podobné jako u regulárních jazyků, kde nám Nerodova věta dává možnost, jak sporem ukázat, že jazyk není regulární. Stejně tak existuje tvrzení – pumping lemma, která formuluje vlastnost, kterou mají BKJ. Jazyky, které nejsou bezkontextové pak tuto vlastnost splňovat nebudou. Například pro jazyk $L=\{0^n1^n2^n;\ n \geq 0\}$ není možné sestrojit bezkontextovou gramatiku, tedy není regulární. Díky této lemmě to však umíme i dokázat.

4.1. Chomského normální forma a pumping lemma

Všechny bezkontextové jazyky lze generovat bezkontextovými gramatikami v jistém speciálním tvaru. Derivační stromy, příslušné k těmto gramatikám, pak mají jednotný tvar.
Tato skutečnost usnadňuje cestu k závěrům v některých úvahách o bezkontextových jazycích.
- Chomského normální forma (Lemma o vkládání)
- Greibachové normální forma

Definice 12: Bezkontextová gramatika je v Chomského normální formě (CHNF), jestliže všechna pravidla jsou v jednom z tvarů:
$X \rightarrow YZ$, $X \rightarrow a$
kde X,Y,Z jsou neterminály a a je terminál.

Chomského NF

Věta 10: Ke každému bezkontextovému jazyku L existuje gramatika G v CHNF taková, že $L(G)=L-\{e\}$.

Důkaz:

Z věty o nevypouštějících gramatikách plyne existence nevypouštějící gramatiky G′ takové, že L(G′)=L – {e}. Ukážeme úpravu pravidel gramatiky G′, která vytvoří žádanou G v CHNF.
Ve třech krocích se zbavíme pravidel porušujících podmínky CHNF.
1. Zbavíme se pravidel typu $X\rightarrow Y$, kde X,Y jsou neterminály, tato pravidla označme jako špatná a ostatní jako dobrá. Pro každý neterminál A zjistěme všechny neterminály B takové, že $A\Rightarrow_{G'}^{*} B$ (tedy $A\Rightarrow Y_1\Rightarrow Y_2\Rightarrow ...\Rightarrow Y_k\Rightarrow B$). Přitom pro každé dobré pravidlo $B\rightarrow\alpha$, zařadíme mezi pravidla i $A\rightarrow\alpha$. Nakonec odstraníme špatná pravidla. Je zřejmé, že generovaný jazyk se nezměnil.
2. Nyní už můžeme předpokládat, že všechna pravidla $A\rightarrow\alpha$, kde $|\alpha| = 1$ jsou "v pořádku" (α je terminál). Pro každé pravidlo:
$A\rightarrow B_1 B_2 B_3... B_n$ $(n \geq 2)$ (10)
kde B_i jsou terminály či neterminály (i=1,2,... n) provedeme následující. Pro každý terminál B_i zavedeme nový neterminál C_i, přidáme pravidlo $C_i\rightarrow B_i$ a v (10) nahradíme B_i tím novým neterminálem C_i.
3. Nyní lze předpokládat všechna pravidla ve tvaru $X\rightarrow$ a nebo $X \rightarrow \alpha$, kde a je terminál a α je řetězec neterminálů délky alespoň 2. Ovšem pravidlo $A\rightarrow C_1 C_2... C_n$ $(n \geq 2)$ lze nahradit soustavou $A\rightarrow C_1 Z_1$, $Z_1\rightarrow C_2 Z_2$, ..., $Z_{n-3}\rightarrow C_{n-2} Z_{n-2}$, $Z_{n-2}\rightarrow C_{n-1} C_n$, kde $Z_1,Z_2,... Z_{n-2}$ jsou nově přidané neterminály.
Lze snadno ověřit, že uvedené úpravy převedou gramatiku do CHNF, přičemž zachovávají generovaný jazyk.

Díky tomuto postupu také můžete převést každou gramatiku do CHNF.

Poznámka: V derivačním stromu podle gramatiky v CHNF má každý vrchol buď dva následníky označené neterminály nebo jednoho následníka označeného terminálem.

Pumping lemma (věta o vkládání, uvwxy – teorém)

Věta 11: (lemma o vkládání, pumping lemma, uvwxy - teorém)
Nechť L je bezkontextový jazyk, pak existují přirozená čísla p,q taková, že každé slovo $z \in L$, které je delší než p ($|z| > p$), se dá psát ve tvaru z=uvwxy přičemž platí následující tři podmínky:
1. $vx \neq e$ (alespoň jedno ze slov v,x je neprázdné)
2. $|vwx| \leq q$
3. $uv^iwx^iy \in L$ pro všechna $i \geq 0$.

Důkaz:
Mějme G=(Π,Σ,S,P) v CHNF, počet neterminálů označme k. Předpokládejme, že v derivačním stromě pro slovo $z \in L$ existují na nějaké cestě od kořene k listu dva různé vrcholy označené týmž neterminálem, řekněme A.

Pak lze snadno ukázat, že $S \Rightarrow^* uAy \Rightarrow^* uvAxy \Rightarrow^* uvwxy$ pro nějaké $u,v,w,x,y \in \Sigma^*$.
Je zřejmé, že buď v nebo x je neprázdné a také, že $uv^i w x^i y \in L, \forall i \geq 0$.
Na každé cestě od kořene k listu derivačního stromu, která je délky alespoň k+1, se nutně vyskytují alespoň dva různé vrcholy označené týmž neterminálem.
Jsou-li v derivačním stromu slova $z \in L$ všechny cesty od kořene k listu kratší než k+1, pak $|z| \leq 2^{k-1}$.
Vezměme nyní $z \in L$ tak, že $|z| > 2^{k-1}$. Nyní na nejdelší cestě od kořene k listu nalezneme dvojici různých vrcholů v_1,v_2 (v_1 je blíž k vrcholu), kde v_1 i v_2 jsou označeny týmž neterminálem a vzdálenost v_1 k listu je nejvýše k+1. Taková dvojice nutně existuje. Pak ovšem "pod v_1" je maximálně 2^k listů. Z uvedených úvah je zřejmé, že lze volit $p=2^{k-1}$, $q=2^k$.
Jelikož naše úvahy platí pro libovolnou gramatiku v CHNF, podle věty má požadované vlastnosti každý jazyk $L-\{e\}$, kde L je libovolný bezkontextový jazyk.
Tvrzení věty se týká pouze slov kladné délky, a tedy z jeho platnosti pro $L-\{e\}$ plyne jeho platnost i pro L.

Pumping lemma nám dává nástroj na odhalování, že jazyk není bezkontextový. Pokud by byl bezkontextový, pak pro něj musí platit podmínka 3., která říká, že lze nalézt takové úseky slova, že když část v a x budeme pumpovat, budou vznikat slova z tohoto jazyka. Alespoň jedna část v nebo x musí být přitom neprázdná (podle 1). Zároveň toto platí pro slova od určité velikosti (konečná množina slov se může i v bezkontextovém jazyce vymykat tomuto pravidlu).

Řešený příklad 9:
Vezměme jazyk $L=\{0^n1^n2^n; n \geq 0\}$ a dokažme, že tento jazyk nemůže být bezkontextový:

Aplikace, význam pumping lemmy

Pokud má být jazyk bezkontextový, platí pro něj pumping lemma. Tedy můžeme najít úseky v a x, které pumpováním vytvářejí slova z jazyka. Rozeberme možnosti, jak mohou tyto úseky vypadat:

v = 01 nebo v = 12 nebo x = 01 nebo x = 12, pak by vznikaly slova ...0101... nebo ...1212..., která jistě nejsou z jazyka, tedy v a u musí obsahovat slova ze stejných symbolů

v = 0, x = 0 (v = 1,x = 1) (v = 2,x = 2), pak ale budou vznikat slova z různým počtem 0,1,2

stejný případ je pokud v = 0, x = 1 nebo v = 0, x = 2 nebo v = 1, x = 2.

Nelze tedy naplnit podmínky lemmy a z toho plyne, že jazyk nemůže být bezkontextový.

4.2. Greibachové normální forma

Greibachové normální forma nabízí tvar, který má své výhody při generování a analýze jazyků. Pokud chceme vygenerovat určité slovo v gramatice, může pro nás být dobré, pokud vidíme, jaké pravidlo použít (lze-li si vybrat z více možností). Proto je rozumné mít tvar pravidel, který začíná terminálem, podle něhož můžeme takové rozhodnutí učinit.

Greibachové NF

Definice 13: Bezkontextová gramatika je v Greibachové normální formě (GNF), právě když jsou všechna pravidla tvaru $X\rightarrow a\alpha$, $X \in \Pi$, $a \in \Sigma$ a α je řetězec neterminálů (popř. prázdný) $\alpha \in \Pi^*$.

Věta 12: Definujme X-pravidlo jako pravidlo, které má na levé straně neterminál X. Nechť
$G=(\Pi,\Sigma,S,P)$ je bezkontextová gramatika, nechť $X\rightarrow\alpha_1 Y\alpha_2$ ($X,Y \in \Pi$, $\alpha_1,\alpha_2 \in (\Pi\cup\Sigma)^*$) je pravidlo v P a nechť $\{Y\rightarrow \beta_1, Y\rightarrow \beta_2, \ldots Y\rightarrow \beta_r\}$ ($\beta_i \in (\Pi\cup\Sigma)^*$, $1 \le i \le r$) je množina všech Y-pravidel.
Nechť $G_1=(\Pi,\Sigma,S,P_1)$ je gramatika získaná z gramatiky G vyloučením pravidla $X\rightarrow\alpha_1 Y\alpha_2$ a přidáním pravidel $X\rightarrow\alpha_1\beta_1\alpha_2$, $X\rightarrow\alpha_1\beta_2\alpha_2,\ldots,X\rightarrow\alpha_1\beta_r\alpha_2$.
Pak $L(G)=L(G_1)$.

Důkaz:
Je zřejmé, že $L(G_1) \subseteq L(G)$, a to proto, že jestliže v G_1 použijeme pravidlo
$X\rightarrow \alpha_1\beta_i\alpha_2$
pak
$X\Rightarrow \alpha_1 Y\alpha_2\Rightarrow \alpha_1\beta_i\alpha_2$
se dá použít v G.
Abychom ukázali, že $L(G) \subseteq L(G_1)$ stačí si jen uvědomit, že jediné pravidlo, které v G_1 v porovnání s G chybí, je pravidlo $X\rightarrow \alpha_1 Y\alpha_2$. Je-li ale v nějakém odvození v gramatice G použito pravidlo $X\rightarrow \alpha_1 Y\alpha_2$, neterminál Y musí být v některém z dalších kroků přepsán použitím pravidla $Y\rightarrow \beta_i$. Tyto dva kroky se dají nahradit jedním krokem $X\Rightarrow_{G1} \alpha_1\beta_i\alpha_2$.

Věta 13: Nechť $G=(\Pi,\Sigma,S,P)$ je bezkontextová gramatika.
Nechť $\{X\rightarrow X\alpha_1, X\rightarrow X\alpha_2, \ldots X\rightarrow X\alpha_r\}$ ($\alpha_i \in (\Pi\cup\Sigma)^*$, $1 \le i \le r$) je množina X-pravidel, ve kterých se na pravé straně úplně vlevo nachází neterminál X. Nechť $\{X\rightarrow \beta_1, X\rightarrow \beta_2, \ldots X\rightarrow \beta_s\}$ ($\beta_i \in (\Pi\cup\Sigma)^*$, $\beta_i \neq X\gamma$, $\gamma \in (\Pi\cup\Sigma)^*$, $1 \le i \le s$) jsou zbývající pravidla mající na levé straně neterminál X. Nechť $G_1=(\Pi\cup\{Z\},\Sigma,S,P_1)$ je bezkontextová gramatika, která vznikla přidáním neterminálu Z k Π a dál nahrazením všech X-pravidel pravidly:
$X\rightarrow\beta_i$, pro $1 \le i \le s$
$X\rightarrow\beta_i Z$, pro $1 \le i \le s$
$Z\rightarrow\alpha_i$, pro $1 \le i \le r$

$Z \to \alpha_i Z$, pro $1 \le i \le r$
Pak $L(G_1)=L(G)$.

Poznámka: Uvědomme si, že všechna X-pravidla gramatiky G generují regulární množinu
$\{\beta_1,\beta_2,...\beta_s\}\{\alpha_1,\alpha_2,...\alpha_r\}^*$
což je právě množina generovaná v G_1 pravidly, která mají na levé straně neterminál X nebo Z.

Důkaz:
Nechť w patří do L(G). Z levého odvození slova w v G můžeme sestrojit odvození w v G_1 takto:
objeví-li se v levém odvození (v G) posloupnost kroků
$uX\gamma \Rightarrow_G uX\alpha_{j1}\gamma \Rightarrow_G uX\alpha_{j2}\alpha_{j1}\gamma \Rightarrow_G ... \Rightarrow_G uX\alpha_{jp}...\alpha_{j2}\alpha_{j1}\gamma \Rightarrow_G u\beta_i\alpha_{jp}...\alpha_{j2}\alpha_{j1}\gamma$
nahradíme tuto posloupnost kroků posloupností
$uX\gamma \Rightarrow_{G1} u\beta_i Z\gamma \Rightarrow_{G1} u\beta_i\alpha_{jp}Z\gamma \Rightarrow_{G1} ... \Rightarrow_{G1} u\beta_i\alpha_{jp}...\alpha_{j2}Z\gamma \Rightarrow_{G1} u\beta_i\alpha_{jp}...\alpha_{j2}\alpha_{j1}\gamma$.
Výsledné odvození je odvození slova w v G_1, ikdyž není levé. Tedy $L(G) \subseteq L(G_1)$.
Vezměme nyní levé odvození slova w v gramatice G_1.
Hned co se v nějakém kroku odvození objeví neterminál Z, změníme uspořádání kroků odvození tak, že hned použijeme pravidla, která způsobí odstranění Z.
Tj. pro nějaké Z se může použít pravidlo $Z \to \alpha Z$. V levém odvození se pak z α odvodí terminální slovo a znovu se použije pravidlo přepisující neterminál Z. Je zřejmé, že α můžeme dočasně nechat tak, a hned použít pravidlo přepisující Z. Pochopitelně toto odvození nebude levé. Nakonec použijeme pravidlo $Z \to \beta$, kde β neobsahuje Z. Vytvořené řetězce α, stejně jako řetězec β, můžeme pak normálně dál přepisovat.
Výsledek tohoto jiným způsobem uskutečněného odvození bude stejný jako u původního levého odvození.
Nahraďme nyní posloupnost kroků obsahujících Z, kterou jsme dostali, tj. nahraďme
$uX\gamma \Rightarrow_{G1} u\beta_i Z\gamma \Rightarrow_{G1} u\beta_i\alpha_{jp}Z\gamma \Rightarrow_{G1} ... \Rightarrow_{G1} u\beta_i\alpha_{jp}...\alpha_{j2}Z\gamma \Rightarrow_{G1} u\beta_i\alpha_{jp}...\alpha_{j2}\alpha_{j1}\gamma$.
posloupností kroků
$uX\gamma \Rightarrow_G uX\alpha_{j1}\gamma \Rightarrow_G uX\alpha_{j2}\alpha_{j1}\gamma \Rightarrow_G ... \Rightarrow_G uX\alpha_{jp}...\alpha_{j2}\alpha_{j1}\gamma \Rightarrow_G u\beta_i\alpha_{jp}...\alpha_{j2}\alpha_{j1}\gamma$.
Výsledkem je odvození slova w v G. Platí $L(G_1) \subseteq L(G)$.

Věta 14: Ke každému bezkontextovému jazyku L existuje gramatika G v GNF taková, že
$L(G)=L - \{e\}$.

Důkaz:
Z věty o Chomského normální formě plyne existence gramatiky G′ v CHNF takové, že $L(G')=L-\{e\}$.
Ukážeme konstrukci G v GNF takové, že $L(G)=L(G')$.
Předpokládejme, že $\Pi = \{X_1,X_2,... X_m\}$.

Prvním krokem
v konstrukci bude úprava pravidel tak, že je-li $X_i \rightarrow X_j\ \gamma$ pravidlo, pak $j > i$. Toto lze provést následujícím způsobem, počínaje X_1 konče X_m. Předpokládejme, že pravidla byla upravena tak, že pro $1 \le i \le k$ je $X_i \rightarrow X_j\ \gamma$ pravidlo pouze je-li $j > i$.
Upravme nyní X_{k+1}-pravidla:
$X_{k+1} \rightarrow X_j\ \gamma$ je takové pravidlo, že $j < k+1$, vytvoříme novou množinu pravidel nahrazením X_j pravou stranou každého jednoho z X_j-pravidel (je-li X_j-pravidel n, pak z pravidla $X_{k+1} \rightarrow X_j\ \gamma$, $(j < k+1)$ vznikne v prvním kroku této úpravy n pravidel) a touto soustavou pravidel nahradíme podle lemmatu 74 pravidlo $X_{k+1} \rightarrow X_j\ \gamma$, $(j < k+1)$. Zopakujeme-li tuto úpravu nejvíce k-krát, dostaneme pravidla ve tvaru $X_{k+1} \rightarrow X_l\gamma$, $l \ge k+1$.
Poté nahradíme pravidla, kde $l=k+1$ podle lemmatu věty předchozí zavedením nové proměnné Z_{k+1}.
Jestliže předcházející postup zopakujeme pro každou původní proměnnou, získáme pravidla, které mají jeden z těchto tvarů:
1. $X_k \rightarrow X_l\gamma$, $l > k$
2. $X_k \rightarrow a\gamma$, $a \in \Sigma$
3. $Z_k \rightarrow \gamma$, $\gamma \in (\Pi \cup \{Z_1, Z_2, \ldots Z_m\})^*$

Druhý krok
Poznamenejme, že levý krajní symbol na pravé straně každého pravidla pro X_m musí být terminál, protože X_m je proměnná s nejvyšším pořadovým číslem. Levý krajní symbol na pravé straně každého $X_{(m-1)}$-pravidla musí být buď X_m nebo terminál. Je-li to X_m můžeme vytvořit nová pravidla tak, že nahradíme X_m pomocí pravých stran pravidel pro X_m ve smyslu lemmatu 74 (tyto pravidla mají pravé strany začínající terminálem).

Poté pokračujeme dál s pravidly pro $X_{(m-2)}$, $X_{(m-3)}$, ... X_2, X_1 až do té doby dokud pravá strana každého pravidla pro nějaké X_i, $(1 \le i \le m)$ nebude začínat terminálem.

Převod na GNF

Třetí krok
Pravidla pro nové neterminály $Z_1, Z_2, \ldots Z_m$ začínají buď terminálem nebo nějakým původním neterminálem. A tedy další použití lemmatu 74 pro každé Z_i-pravidlo $(1 \le i \le m)$ ukončuje celou konstrukci.

Řešený příklad 10:

Sestrojte k následující bezkontextové gramatice v CHNF ekvivalentní bezkontextovou gramatiku v GNF.
$A_1 \rightarrow A_2A_3$
$A_2 \rightarrow A_3A_1|b$
$A_3 \rightarrow A_1A_2|a$

1.3.1.
$A_1 \rightarrow A_2A_3$
$A_2 \rightarrow A_3A_1|b$
$A_3 \rightarrow A_2A_3A_2|a$

1.3.2.
$A_1 \rightarrow A_2A_3$
$A_2 \rightarrow A_3A_1|b$
$A_3 \rightarrow A_3A_1A_3A_2|bA_3A_2|a$
1.3.3.
$A_1 \rightarrow A_2A_3$
$A_2 \rightarrow A_3A_1|b$
$A_3 \rightarrow bA_3A_2|a|bA_3A_2Z_3|aZ_3$
$Z_3 \rightarrow A_1A_3A_2|A_1A_3A_2Z_3$
2.2.
$A_1 \rightarrow A_2A_3$
$A_2 \rightarrow bA_3A_2A_1|aA_1|bA_3A_2Z_3A_1|aZ_3A_1|b$
$A_3 \rightarrow bA_3A_2|a|bA_3A_2Z_3|aZ_3$
$Z_3 \rightarrow A_1A_3A_2|A_1A_3A_2Z_3$
2.1.
$A_1 \rightarrow bA_3A_2A_1A_3|aA_1A_3|bA_3A_2Z_3A_1A_3|aZ_3A_1A_3|bA_3$
$A_2 \rightarrow bA_3A_2A_1|aA_1|bA_3A_2Z_3A_1|aZ_3A_1|b$
$A_3 \rightarrow bA_3A_2|a|bA_3A_2Z_3|aZ_3$
$Z_3 \rightarrow A_1A_3A_2|A_1A_3A_2Z_3$
3.3.
$A_1 \rightarrow bA_3A_2A_1A_3|aA_1A_3|bA_3A_2Z_3A_1A_3|aZ_3A_1A_3|bA_3$
$A_2 \rightarrow bA_3A_2A_1|aA_1|bA_3A_2Z_3A_1|aZ_3A_1|b$
$A_3 \rightarrow bA_3A_2|a|bA_3A_2Z_3|aZ_3$
$Z_3 \rightarrow$
$bA_3A_2A_1A_3A_3A_2|aA_1A_3A_3A_2|bA_3A_2Z_3A_1A_3A_3A_2|aZ_3A_1A_3A_3A_2|aA_3A_3A_2$
$Z_3 \rightarrow$
$bA_3A_2A_1A_3A_3A_2Z_3|aA_1A_3A_3A_2Z_3|bA_3A_2Z_3A_1A_3A_3A_2Z_3|aZ_3A_1A_3A_3A_2Z_3|a$
$A_3A_3A_2Z_3$

Řešený příklad 11:

K bezkontextové gramatice G sestrojte ekvivalentní (popřípadě až na {e}) bezkontextovou gramatiku G_1 v Chomského normální formě.

a) G:$S \rightarrow ABC$, $S \rightarrow BC$,$A \rightarrow 011A$, $A \rightarrow 011$, $B \rightarrow 10$, $C \rightarrow 1C0$, $C \rightarrow 0$.

Řešení:
G_p:$S \rightarrow ABC$, $S \rightarrow BC$,$A \rightarrow NJJA$, $A \rightarrow NJJ$, $B \rightarrow JN$, $C \rightarrow JCN$, $C \rightarrow 0$, $N \rightarrow 0$, $J \rightarrow 1$.
G_1:$S \rightarrow AA_1$, $A_1 \rightarrow BC$, $S \rightarrow BC$,$A \rightarrow NA_2$, $A_2 \rightarrow JA_3$, $A_3 \rightarrow JA$, $A \rightarrow NA_4$, $A_4 \rightarrow JJ$, $B \rightarrow JN$, $C \rightarrow JA_5$, $C \rightarrow 0$, $A_5 \rightarrow CN$, $N \rightarrow 0$, $J \rightarrow 1$.

b) G:$S \rightarrow A$, $S \rightarrow B$,$A \rightarrow abaC$, $A \rightarrow aba$, $B \rightarrow D$, $C \rightarrow bC$, $C \rightarrow b$, $D \rightarrow bbD$, $D \rightarrow aE$, $D \rightarrow a$, $E \rightarrow D$.

Řešení:
Sestrojíme relaci čtvereček na množině Π a její reflexivní tranzitivní uzávěr:
čtv:{(S,A),(S,B),(B,D),(E,D)}
uzávěr čtv:{(S,S),(A,A),(B,B),(C,C),(D,D),(E,E),(S,A),(S,B),
(B,D),(E,D),(S,D)}
G_p:A→ abaC, A→ aba, C→ bC, C→ b, D→ bbD, D→ aE, D→ a, S→ abaC, S→ aba, B→ bbD, B→ aE, B→ a,E→ bbD, E→ aE, E→ a,S→ bbD, S→ aE, S→ a.
G_{pom}:A→ XYXC, A→ XYX, C→ YC, C→ b, D→ YYD, D→ XE, D→ a, S→ XYXC, S→ XYX, B→ YYD, B→ XE, B→ a,E→ YYD, E→ XE, E→ a,S→ YYD, S→ XE, S→ a, X→ a, Y→ b.
G_1:A→ XA_1, A_1→ YA_2, A_2→ XC,A→ XA_3, A_3→ YX, C→ YC, C→ b, D→ YA_4, A_4→ YD, D→ XE, D→ a, S→ XA_5, A_5→ YA_6, A_6→ XC, S→ XA_7, A_7→ YX, B→ YA_8, A_8→ YD, B→ XE, B→ a,E→ YA_8, E→ XE, E→ a,S→ YA_8, S→ XE, S→ a, X→ a, Y→ b.

Kontrolní otázka:

K bezkontextové gramatice G sestrojte ekvivalentní bezkontextovou gramatiku G_1 v Greibachové normální formě.
G:S→ XY, S→ YZ,X→ XY, X→ a, Y→ XY, Y→ YZ, Y→ b, Z→ c.

Řešení:
1. S < X < Y < Z
2. S→ XY, S→ YZ
X→ a, X→ aZ_x, Z_x→ Y, Z_x→ YZ_x
Převedu
Y→ aY, Y→ aZ_xY, Y→ YZ, Y→ b
a množinu těchto Y-pravidel nahradím množinou pravidel:
Y→ aY, Y→ aZ_xY, Y→ b, Y→ aYZ_y, Y→ aZ_xYZ_y, Y→ bZ_y,Z_y→ Z, Z_y→ ZZ_y
Z→ c.
3.,4., což je výsledná bezkontextová gramatika G_1
Z→ c
Y→ aY, Y→ aZ_xY, Y→ b, Y→ aYZ_y, Y→ aZ_xYZ_y, Y→ bZ_y
X→ a, X→ aZ_x
S→ aY, S→ aZ_xY,S→ aYZ, S→ aZ_xYZ, S→ bZ, S→ aYZ_yZ, S→ aZ_xYZ_yZ, S→ bZ_yZ
Z_x→ aY, Z_x→ aZ_xY, Z_x→ b, Z_x→ aYZ_y, Z_x→ aZ_xYZ_y, Z_x→ bZ_y
Z_x→ aYZ_x, Z_x→ aZ_xYZ_x, Z_x→ bZ_x, Z_x→ aY$Z_y$$Z_x$, Z_x→ aZ_xY$Z_y$$Z_x$, Z_x→ b$Z_y$$Z_x$
Z_y→ c, Z_y→ cZ_y

Nejdůležitější probrané pojmy:

- Chomského normální forma
- Pumping lemma
- Greibachové normální forma

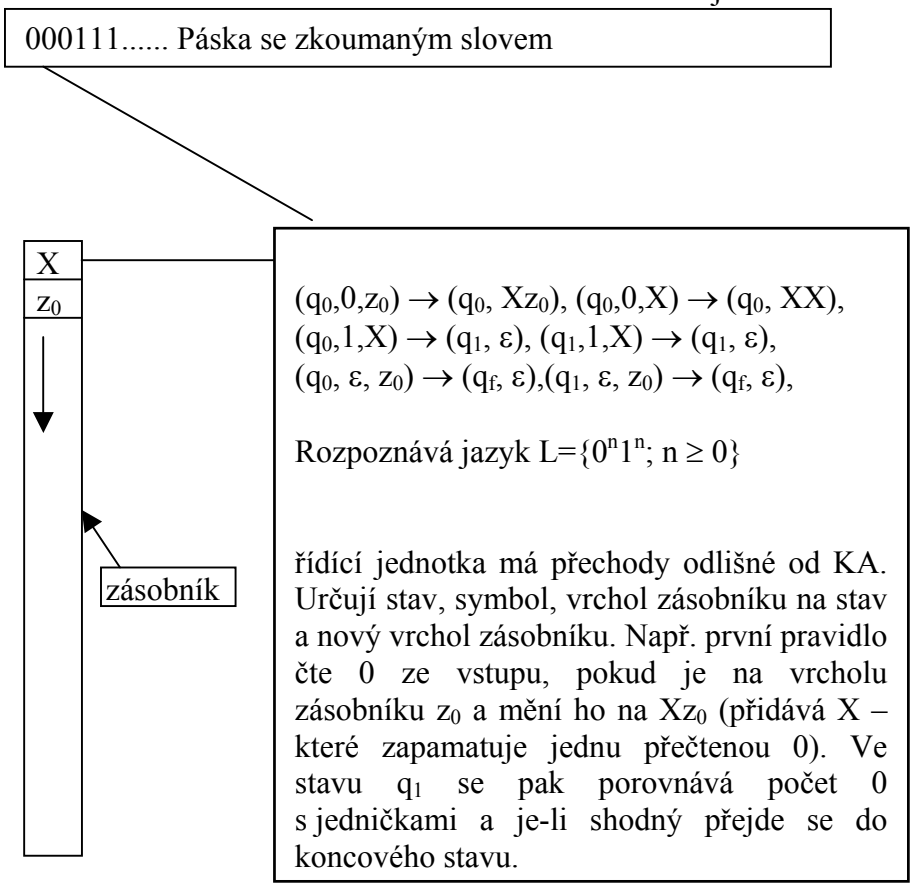
5. Zásobníkové automaty

Cíl:

Po prostudování této kapitoly pochopíte:
- co je zásobníkový automat
- jaký jeho vztah k bezkontextovým jazykům
- co je pumping lemma (lemma o vkládání)

Naučíte se:
- vytvářet zásobníkové automaty pro zadané jazyky

Zásobníkový automat je stroj, který stejně jako konečný automat má nějakou řídící jednotku, která je vždy v nějakém ze svých stavů, a který čte ze vstupní pásky slovo (nad nějakou abecedou) a po jeho přečtení rozhodne, zda slovo patří či nepatří do jazyka, který zásobníkový automat rozpoznává. Avšak narozdíl od konečných automatů, zásobníkový automat využívá navíc zásobníku, neboli jakési paměti typu LIFO. Tedy může ukládat a vybírat symboly na vrchol zásobníku, který si lze představit jako naskládané talíře – nelze je brát odkudkoliv – pouze z vrcholu.
Idea zásobníkového automat se dá znázornit na následujícím obrázku.



5.1. Zásobníkový automat a vztah k BKJ

Zásobníkový automat

Definice 14: Zásobníkovým automatem nazveme sedmici (systém určený sedmi parametry)
$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, kde Q je konečná neprázdná množina stavů, Σ je konečná neprázdná množina vstupních symbolů (abeceda), Γ je konečná neprázdná množina zásobníkových symbolů, $q_0 \in Q$ je počáteční stav, $Z_0 \in \Gamma$ je počáteční zásobníkový symbol, $F \subseteq Q$ je množina koncových stavů a δ je zobrazení množiny $Q\times(\Sigma\cup\{e\})\times\Gamma$ do množiny konečných podmnožin množiny $Q \times\Gamma^*$ (přechodová funkce). ($\delta: Q\times(\Sigma\cup\{e\})\times\Gamma \rightarrow P(Q\times\Gamma^*)$)

Z definice je patrné, že takto definovaný zásobníkový automat je nedeterministický.

Neformálně význam δ (tj. předpisu chování ZA *M*):
Je-li $\delta(q,a,X) = \{(q_1,\alpha_1),(q_2, \alpha_2), ...,(q_n, \alpha_n)\}$; $q \in Q$, $a \in (\Sigma\cup\{e\})$, $q_i \in Q, \alpha_i \in \Gamma^*$, $i \in \{1, 2, ..., n\}$,
$X \in \Gamma$, potom když *M* má čtecí hlavu na symbolu a, (konečná ŘJ) je ve stavu q a na vrcholu zásobníku je symbol X, může si *M* vybrat jedno i z {1, 2, ..., n } a posunout čtecí hlavu o jeden symbol vpravo, změnit stav řídící jednotky na q_i a symbol X v zásobníku nahradit řetězcem α_i. Speciálně je-li a=e, může *M* provést tzv. e-krok, při kterém nečte a hlava se tudíž neposunuje. Říkáme také, že *M* provedl instrukci (q,a,X) $[(\delta) \parallel (\rightarrow)]$ (q_i,α_i).
Důležitá je i skutečnost, že mohou existovat $q \in Q$, $a \in (\Sigma\cup\{e\})$, $X \in \Gamma$ tak, že $\delta(q,a,X)=\varnothing$ (v jistých situacích tedy nemůže automat pokračovat ve výpočtu). Při definici konkrétní přechodové funkce budeme definici obrazu pro takovéto vzory ((q,a,X)) vynechávat.

Konfigurace

Definice 15: Mějme ZA $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$. Situací (konfigurací) zásobníkového automatu *M* nazveme trojici (q, w, α), kde $q \in Q$, $w \in \Sigma^*$ a $\alpha \in \Gamma^*$. q je stav ŘJ, w je slovo (ta část slova) na vstupní pásce, která zbývá přečíst, α je obsah zásobníku. (Nejlevější symbol v α představuje *vrchol zásobníku*). Jestliže $(q', \alpha) \in \delta(q,a,X)$, pak pro lib. $w \in \Sigma^*$, $\beta \in \Gamma^*$ vede situace $(q, aw, X\beta)$ bezprostředně k situaci $(q',w,\alpha\beta)$, symbolicky značíme:
$(q,aw,X\beta) \Rightarrow (q',w,\alpha\beta)$
Nechť E a E′ jsou situace ZA *M*, pak řekneme, že E vede k situaci E′, značíme $E \Rightarrow^* E'$, jestliže existují situace $E_1, E_2, ..., E_n$ tak, že $E=E_1\Rightarrow E_2\Rightarrow...\Rightarrow E_n=E'$. Je-li potřeba, značíme o jaký ZA se jedná:
$\Rightarrow_M$ $\Rightarrow_M^*$

Narozdíl od konečného automatu může ZA rozpoznávat slova nejen tím, že skončí v koncovém stavu, ale také tím, že vyprázdní celý svůj zásobník. Například ilustrace na počátku kapitoly rozpoznává daný jazyk jak prázdným zásobníkem, tak i koncovým stavem.

Rozpoznávání jazyka zásobníkovým automatem budeme definovat dvěma způsoby:
1) přijímání koncovým stavem: slovo w je přijato ZA, jestliže existuje možnost, že po zpracování (přečtení) slova w se automat ocitne v koncovém stavu.
2) přijímání prázdným zásobníkem: slovo w je přijato ZA, jestliže existuje možnost, že po zpracování slova w se ZA ocitne v situaci s prázdným zásobníkem.

Rozpoznávání - koncovým stavem a prázdným zásobníkem

Definice 16: Mějme ZA $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$. Definujme
$L_{KS}(M)=\{w \in \Sigma^* | (q_0, w, Z_0) \Rightarrow_M^* (q,e,\alpha)$ pro nějaké $q \in F$ a $\alpha \in \Gamma^*\}$.
$L_{PZ}(M)=\{w \in \Sigma^* | (q_0, w, Z_0) \Rightarrow_M^* (q,e,e)$ pro libovolné $q \in Q\}$.

Deterministický ZA

Definice 17: ZA $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$ nazveme deterministický (DZA), jestliže platí následující dvě podmínky:
1. $\delta(q, a, X)$ je nejvýše jednoprvková množina pro lib. $q \in Q$, $a \in (\Sigma \cup \{e\})$, $X \in \Gamma$.
2. Jestliže $\delta(q,e,X) \neq \emptyset$ pro něj. $q \in Q$, $X \in \Gamma$, pak $\delta(q,a,X)=\emptyset$ pro lib. $a \in \Sigma$.

Definice 18: Jazyky rozpoznatelné DZA koncovým stavem nazveme deterministické (třídu těchto jazyků označíme Det). Jazyky rozpoznatelné DZA prázdným zásobníkem nazveme bezprefixové deterministické (třídu těchto jazyků označíme BDet).

Poznámka: Dá se ukázat, že Det je vlastní podtřída třídy bezkontextových jazyků.

(Např. jazyk $\{ww^R | w \in \{a,b\}^*\}$ není deterministický.)
(Srovnejte se situací u konečných automatů).

Řešený příklad 12:

Sestrojte zásobníkový automat, který rozpoznává jazyk $L=\{w(w)^R;\ w \in \{0,1\}^*\}$ (prázdným zásobníkem).

Hledaný automat $M=(\{p,q\},\{0,1\},\{A,B,C\},\delta,p,A,\varnothing)$ má přechodovou funkci δ definovánu takto:
$\delta(p,0,A)=\{(p,BA)\}$,
$\delta(p,1,A)=\{(p,CA)\}$,
$\delta(p,0,B)=\{(p,BB),(q,e)\}$,
$\delta(p,0,C)=\{(p,BC)\}$,
$\delta(p,1,B)=\{(p,CB)\}$,
$\delta(p,1,C)=\{(p,CC),(q,e)\}$,
$\delta(q,0,B)=\{(q,e)\}$,
$\delta(q,1,C)=\{(q,e)\}$,
$\delta(p,e,A)=\{(q,e)\}$,
$\delta(q,e,A)=\{(q,e)\}$.

Automat pracuje tak, že za každý symbol ze slova w přidá na zásobník zástupce, který pak porovná v zrcadlovém slově. Jelikož zásobník odebírá z vrcholu symboly rovněž zrcadlově, rozpozná právě slova z jazyk. Ale například jazyk $L=\{ww;\ w \in \{0,1\}^*\}$ (zdvojené slovo) už není možné rozpoznat ZA!

Řešený příklad 13:

Sestrojte zásobníkový automat, který rozpoznává jazyk $L=\{wc(w)^R;\ w \in \{0,1\}^*\}$ (prázdným zásobníkem).

Hledaný automat $M=(\{p,q\},\{0,1\},\{A,B,C\},\delta,p,A,\varnothing)$ má přechodovou funkci δ definovánu takto:
$\delta(p,0,A)=\{(p,BA)\}$,
$\delta(p,1,A)=\{(p,CA)\}$,
$\delta(p,0,B)=\{(p,BB)\}$,
$\delta(p,0,C)=\{(p,BC)\}$,
$\delta(p,1,B)=\{(p,CB)\}$,
$\delta(p,1,C)=\{(p,CC)\}$,
$\delta(p,c,A)=\{(q,e)\}$,
$\delta(p,c,B)=\{(q,B)\}$,
$\delta(p,c,C)=\{(q,C)\}$,
$\delta(q,0,B)=\{(q,e)\}$,
$\delta(q,1,C)=\{(q,e)\}$,
$\delta(q,e,A)=\{(q,e)\}$.

Poznámka: Tento automat je deterministický. Toto je příklad jazyka, ke kterému lze sestrojit DZA. Nicméně jsou i jazyky, ke kterým nelze DZA sestrojit (viz příklad předchozí).

Následující věta nám říká, že rozpoznávání KS a PZ jsou dvě ekvivalentní podmínky. Ke každému ZA KS lze sestrojit ekvivalentní ZA PZ a naopak. U PZ stačí doplnit instrukci, která při prázdném zásobníku automat dostane do koncového stavu. U KS je třeba doplnit více instrukcí, které v koncovém stavu (který změníme na nekoncový), vyprázdní postupně celý zásobník.

Ekvivalence rozpoznávání

Věta 15: Mějme libovolný jazyk L. Pak $L=L_{KS}(M_1)$ pro nějaký ZA M_1, právě když $L=L_{PZ}(M_2)$ pro nějaký ZA M_2.

Nyní formulujeme a dokážeme velmi důležité tvrzení, že jazyky rozpoznávané ZA jsou právě jazyky bezkontextové. Jde o stejný typ tvrzení jako, když jazyky generované regulárními gramatikami byly právě rozpoznatelné konečnými automaty.
Lze také jednoduše sestrojit ke každé gramatice ZA, který bude rozpoznávat generovaný jazyk a to pomocí simulace odvození v gramatice na zásobníku. Zpětně lze každý automat reprezentovat pomocí BKG – složitěji pomocí postupné simulace přechodů mezi stavy ZA

Vztah jazyků rozpoznatelných ZA a BKJ

Věta 16: Ke každému bezkontextovému jazyku L existuje ZA M takový, že $L=L_{PZ}(M)$. Navíc M má jediný stav.

Důkaz:
Mějme bezkontextovou gramatiku $G=(\Pi,\Sigma,S,P)$.
Sestrojíme ZA M tak, že $L(G)=L_{PZ}(M)$.
Položíme $M = (\{p\},\Sigma,\Pi\cup\Sigma,\delta,p,S,\varnothing)$.
Pro δ platí:
$\delta(p,e,X)=\{(p,\alpha)|(X\rightarrow\alpha)\in P\};\ \forall X\in\Pi$
$\delta(p,a,a)=\{(p,e)\};\ \forall a\in\Sigma$

Takto sestrojený ZA má dva typy pravidel – buď přepisuje neterminál na řetězec nebo srovnává terminální symboly. Pokud symboly nesedí, pak se automat zasekne. Obecně je automat nedeterministický – tedy musí si najít správnou cestu.

Věta 17: K libovolnému ZA M s jedním stavem, lze zkonstruovat bezkontextovou gramatiku G tak, že $L_{PZ}(M)=L(G)$.

Věta 18: **K libovolnému ZA M lze zkonstruovat ZA M' s jedním stavem takový, že $L_{PZ}(M)=L_{PZ}(M')$.**

Řešený příklad 14:

Mějme zásobníkový automat $M = (Q=\{p,q\},\Sigma = \{0,1\},\Gamma = \{A,B\},\delta,p,A,\varnothing)$
δ:
δ(p,0,A)={(p,BA)(q,A)}
δ(p,0,B)={(p,BB)}
δ(p,1,B)={(p,e)}
δ(q,0,A)={(q,A)(q,e)}
δ(q,e,A)={(p,BB)}
Zkonstruujte M' s jedním stavem tak, aby $L_{PZ}(M)=L_{PZ}(M')$.
Řešení:
δ′(p,1,⟨p,B,p⟩) ə (p,e)
δ′(p,0,⟨q,A,q⟩) ə (p,e)
δ′(p,0,⟨p,A,p⟩) ə (p,⟨q,A,p⟩)
δ′(p,0,⟨p,A,q⟩) ə (p,⟨q,A,q⟩)
δ′(p,0,⟨q,A,p⟩) ə (p,⟨q,A,p⟩)
δ′(p,0,⟨q,A,q⟩) ə (p,⟨q,A,q⟩)
δ′(p,0,⟨p,A,p⟩) ə (p,⟨p,B,p⟩⟨p,A,p⟩)
δ′(p,0,⟨p,A,p⟩) ə (p,⟨p,B,q⟩⟨q,A,p⟩)
δ′(p,0,⟨p,A,q⟩) ə (p,⟨p,B,p⟩⟨p,A,q⟩)
δ′(p,0,⟨p,A,q⟩) ə (p,⟨p,B,q⟩⟨q,A,q⟩)
δ′(p,0,⟨p,B,p⟩) ə (p,⟨p,B,p⟩⟨p,B,p⟩)
δ′(p,0,⟨p,B,p⟩) ə (p,⟨p,B,q⟩⟨q,B,p⟩)
δ′(p,0,⟨p,B,q⟩) ə (p,⟨p,B,p⟩⟨p,B,q⟩)
δ′(p,0,⟨p,B,q⟩) ə (p,⟨p,B,q⟩⟨q,B,q⟩)
δ′(p,e,⟨q,A,p⟩) ə (p,⟨p,B,p⟩⟨p,B,p⟩)
δ′(p,e,⟨q,A,p⟩) ə (p,⟨p,B,q⟩⟨q,B,p⟩)
δ′(p,e,⟨q,A,q⟩) ə (p,⟨p,B,p⟩⟨p,B,q⟩)
δ′(p,e,⟨q,A,q⟩) ə (p,⟨p,B,q⟩⟨q,B,q⟩)
δ′(p,e,R)={(p,⟨p,A,p⟩),(p,⟨p,A,q⟩)}

Přechodová funkce δ′ zkonstruovaného ZA M' s jedním stavem
δ′(p,e,R)={(p,⟨p,A,p⟩),(p,⟨p,A,q⟩)}
δ′(p,0,⟨p,A,p⟩)={(p,⟨q,A,p⟩),(p,⟨p,B,p⟩⟨p,A,p⟩),(p,⟨p,B,q⟩⟨q,A,p⟩)}
δ′(p,0,⟨p,A,q⟩)={(p,⟨q,A,q⟩),(p,⟨p,B,p⟩⟨p,A,q⟩),(p,⟨p,B,q⟩⟨q,A,q⟩)}
δ′(p,0,⟨q,A,p⟩)={(p,⟨q,A,p⟩)}
δ′(p,0,⟨q,A,q⟩)={(p,e),(p,⟨q,A,q⟩)}
δ′(p,1,⟨p,B,p⟩)={(p,e)}
δ′(p,0,⟨p,B,p⟩)={(p,⟨p,B,p⟩⟨p,B,p⟩),(p,⟨p,B,q⟩⟨q,B,p⟩)}
δ′(p,0,⟨p,B,q⟩)={(p,⟨p,B,p⟩⟨p,B,q⟩),(p,⟨p,B,q⟩⟨q,B,q⟩)}
δ′(p,e,⟨q,A,p⟩)={(p,⟨p,B,p⟩⟨p,B,p⟩),(p,⟨p,B,q⟩⟨q,B,p⟩)}

$\delta'(p,e,\langle q,A,q\rangle)=\{(p,\langle p,B,p\rangle\langle p,B,q\rangle),(p,\langle p,B,q\rangle\langle q,B,q\rangle)\}$

Důsledky vztahů mezi ZA a BKG

Věta 19: (důsledek) Pro libovolný jazyk L jsou následující podmínky ekvivalentní:

1) L je bezkontextový
2) L je rozpoznatelný ZA koncovým stavem
3) L je rozpoznatelný ZA prázdným zásobníkem
4) L je rozpoznatelný ZA s jedním stavem prázdným zásobníkem

Nejdůležitější probrané pojmy:

- zásobníkový automat
- jazyky rozpoznatelné ZA
- vztah k bezkontextovým jazykům
- důsledky těchto vztahů

Kontrolní otázka:

Existují jazyky rozpoznatelné ZA, které nejsou rozpoznatelné deterministickým ZA?

Řešení:

Takový jazyk existuje (uvedený v předchozím textu).

Korespondenční úkol:

Část 1:
Vyberte si dva bezkontextové jazyky, ke kterým nebyl v tomto textu sestrojen ZA a sestrojte jej. Pokud to jde, sestrojte DZA.

Část 2:
Sestrojte gramatiky k vybraným jazykům z části 1., které budou v CHNF a GNF.

6. Vlastnosti tříd bezkontextových jazyků

Cíl:

Po prostudování této kapitoly pochopíte:
- uzávěrové vlastnosti BKJ

Naučíte se:
- Parikhovu větu

Stejně jako u regulárních jazyků lze sledovat, zda je třída BKJ uzavřena na množinové a jiné operace. V tomto případě to nebude platit pro všechny operace. Dále se naučíte Parikhovu větu, která je alternativní možností, jak dokazovat, že jazyky nejsou bezkontextové.

6.1. Uzávěrové vlastnosti třídy BKJ

Třída bezkontextových jazyků je uzavřena především vůči sjednocení, zřetězení a iteraci. Je velice jednoduché sestrojit k gramatikám jejich sjednocení a další operace.

Mějme $G_1=(\Pi_1,\Sigma,S_1,P_1)$ a $G=(\Pi_2,\Sigma,S_2,P_2)$ zkonstruovat gramatiky k jazykům $L_1=L(G_1)$ a $L_1=L(G_2)$ po aplikaci uzávěrových operací lze následovně:

$L_1 \cup L_2$: G: $S \rightarrow S_1$, $S \rightarrow S_2$, + P_1 + P_2 *(tedy vygeneruje slovo podle G_1 nebo G_2)*

$L_1 \cdot L_2$: G: $S \rightarrow S_1S_2$, + P_1 + P_2 *(tedy vygeneruje slovo podle G_1 a za ním podle G_2)*

L_1*: G: $S \rightarrow S_1S$, $S \rightarrow \varepsilon$, + P_1 *(tedy vygeneruje libovolněkrát slovo podle G_1)*

Uzávěrové vlastnosti

Věta 20: Třída bezkontextových jazyků je uzavřena vůči: sjednocení, zřetězení, iteraci, zrcadlovému obrazu, homomorfismu a substituci. (Ale je také uzavřena např. vůči průniku s regulárním jazykem a vůči kvocientu podle regulárního jazyka).

Průnik a doplněk však nemohou být sestrojeny, protože třída BKJ není uzavřena vůči těmto operacím. Existují totiž BKJ, jejichž průnik není BKJ. Příkladem budiž jazyk:

$L_1=\{0^n1^n2^m; n,m \geq 0\}$ a $L_2=\{0^m1^n2^n; n,m \geq 0\}$, pak
$L_1 \cap L_2 = \{0^n1^n2^n; n \geq 0\}$, který ovšem jak už víme, není bezkontextový.

Věta 21: Třída bezkontextových jazyků není uzavřena vůči průniku a doplňku.

Alternativním způsobem, jak dokazovat, že jazyk není bezkontextový je Parikhova věta. Považujte ji prosím spíše za doplňující učivo. Není vyžadováno ke zkoušce.

Definice 19: Nechť L je jazyk nad abecedou Σ ($L \subseteq \Sigma^*$), $\Sigma = \{a_1,a_2,a_3,... a_n\}$, $w \in L$.
Parikhův vektor ψ(w) slova w definujeme takto: $\psi(w)=(a_1, a_2, ... , a_n)$, i-tá složka a_i vektoru ψ(w) je počet písmen a_i ve slově w.
Množinu Parikhových vektorů ψ(L) jazyka L definujeme takto: $\psi(L)=\{\psi(w)|w \in L\}$

Poznámka: Jsou-li $\alpha_0,\alpha_1,...,\alpha_m \in N^n$ (neboli n-rozměrné vektory), pak lineární podmnožinou tvořenou prvky α budeme rozumět takovouto množinu:
$\{\alpha_0+n_1\alpha_1+... +n_m\alpha_m|n_j \geq 0, 1 \leq j \leq m\}$
a dále semilineární množinou budeme rozumět konečné sjednocení lineárních podmnožin.

Věta 22: Je-li L bezkontextový jazyk, pak existuje regulární jazyk L_R tak, že $\psi(L)=\psi(L_R)$. Navíc $\psi(L_R)$ je semilineární pro všechny regulární jazyky.

Nejdůležitější probrané pojmy:

- uzávěrové vlastnosti třídy BKJ
- Parikhova věta

Takový jazyk existovat nemůže, neboť známe postup jak každý NKA převést na DKA.

Úkol k textu:

Vezměte si libovolné dva BKJ z tohoto textu a sestrojte gramatiku pro jejich sjednocení a zřetězení.

7. Chomského hierarchie

Cíl:

Po prostudování této kapitoly pochopíte:
- Hierarchizaci jazyků v jejich obecnosti
- Které jazyky jsou složitější než regulární a bezkontextové
- Jak pracují automaty pro složitější jazyky

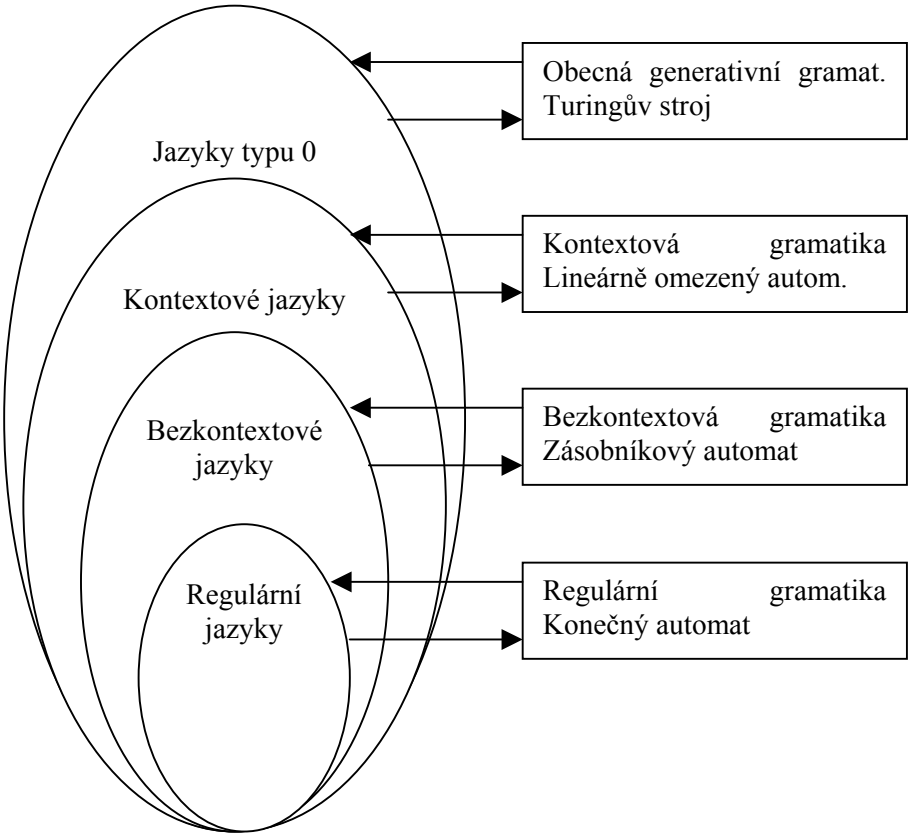
Naučíte se:

- Klasifikovat jazyky z hlediska Chomského hierarchie

Během vašeho studia teorie formálních jazyků jste se seznámili především se dvěmi třídami jazyků – regulárními a bezkontextovými. Existují ale samozřejmě i vyšší třídy jazyků (složitější). Vzpomeňte si na obecný pojem gramatiky. Právě tyto obecné gramatiky generují nejvyšší třídu jazyků (tzv. jazyky typu 0) podle Chomského hierachie. Právě podle již zmiňovaného Noama Chomského se tato klasifikace jazyků, podle toho jaké typy gramatik je generují, nazývá.
Na obrázku můžete toto rozdělení vidět. Chomského hierarchie obsahuje 4 třídy jazyků, které lze generovat generativními gramatikami. Samozřejmě, že s použitím generativních gramatik nelze vytvořit všechny jazyky – tyto jazyky jsou pak nad touto hierarchií. Pro teoretické výsledky teorie vyčíslitelnosti je důležitá třída jazyků typy 0 a kontextové jazyky (typu 1). Jazyky kontextové mají navíc význam pro umělou inteligenci, konkrétně analýzu přirozeného jazyka. Pro aplikované oblasti informatiky mají význam především jazyky bezkontextové (typu 2) a regulární (typu 3) a to při definování struktur programovacích a jiných jazyků používaných v praxi. Kromě gramatiky je důležitý zmíněný duální pojem automatu, který rozpoznává slova jazyka. Na obrázku jsou také ke každé třídě připojeny příslušné duální pojmy gramatiky - automatu.
V Chomského hierarchii je možné dále rozlišovat podtřídy podle toho zda jazyky lze analyzovat pomocí deterministického nebo nedeterministického automatu. Zvláště důležité to je pro třídu bezkontextových jazyků, které korespondují s používanými programovacími jazyky. Deterministické jazyky (rozpoznatelné deterministickými zásobníkovými automaty) jsou ve svých speciálních formách jako LL nebo LR jazyky efektivně analyzovatelné. Existují i alternativní hierarchie jazyků založené na odlišných přístupech ke generování jazyků, z nichž zřejmě nejznámější jsou Lindenmayerovy systémy využívané například v biologii pro simulaci chování živých organismů. Teorie jazyků je důležitou součástí informatiky a její poznatky se aplikují nejen v informatice samotné.

7.1. Obecná generativní gramatika a Chomského hierarchie

Chomského hierarchie



Generativní gramatika

Definice 20: Generativní gramatika je čtveřice $G=(\Pi,\Sigma,S,P)$, kde všechny parametry mají tentýž význam jako u bezkontextových gramatik s tím, že přepisovací pravidla jsou obecně tvaru $\alpha\rightarrow\beta$, kde $\alpha,\beta \in (\Pi\cup\Sigma)^*$, přičemž α obsahuje alespoň jeden neterminál.
Řekneme, že γ se přímo přepíše na δ a značíme $\gamma\Rightarrow\delta(\gamma,\delta \in (\Pi\cup\Sigma)^*)$, jestliže lze psát $\gamma = \gamma_1\alpha\gamma_2$, $\delta = \gamma_1\beta\gamma_2$, kde $(\alpha\rightarrow\beta) \in P$.
Relace $\Rightarrow^*$ je reflexivní a tranzitivní uzávěr relace $\Rightarrow$.
Jazyk generovaný gramatikou G je $L(G)=\{w \in \Sigma^* | S\Rightarrow^* w\}$.

Nejstarší a nejznámější hierarchie gramatik podle tvarů přepisovacích pravidel je tzv. Chomského hierarchie.

Definice 21: Generativní gramatika $G=(\Pi,\Sigma,S,P)$ je
0) typu 0, jestliže na pravidla neklademe žádná omezení
1) typu 1, neboli kontextová gramatika, jestliže všechna pravidla jsou ve tvaru $\alpha X\beta\rightarrow\alpha\gamma\beta$, kde $|\gamma| \geq 1$, $\alpha,\beta,\gamma \in (\Pi\cup\Sigma)^*$, $X \in \Pi$. Jedinou vyjimkou je pravidlo typu $S\rightarrow e$, které se v gramatice objevit může, v tom případě se ale S nesmí objevit na pravé straně žádného pravidla.
2) typu 2, neboli bezkontextová gramatika (viz. dřívější definice)
3) typu 3, neboli regulární gramatika (viz. dřívější definice)

Věta 23: **Nechť L_i označuje třídu jazyků typu i. Pak $L_3 \subset L_2 \subset L_1 \subset L_0$.**

Důkaz:
$L_3 \subset L_2$, $L_1 \subset L_0$ triviálně platí.
$L_2 \subset L_1$ řeší se pomocí nevypouštějících bezkontextových gramatik.
Všechny inkluze jsou vlastní.
Např. $\{a^n b^n\} \in (L_2 - L_3)$.
Dále $\{a^n b^n c^n\} \in (L_1 - L_2)$. (viz následující příklad).
Inkluzi $L_1 \subset L_0$ nyní řešit nebudeme.

Třída kontextových jazyků, jak ji vidíte v definici obsahuje také jazyk, o kterém jsme dříve dokázali, že není bezkontextový. Kontextové gramatiky přepisují neterminály také v **kontextu** dalších slov. Nejlépe je to vidět na gramatice pro zmíněný jazyk.

Řešený příklad 15:

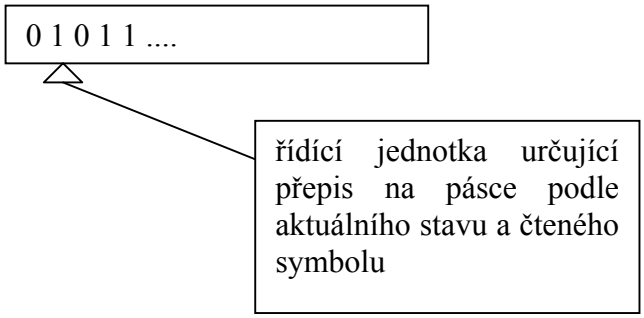
Příklad: Gramatika pro $L=\{a^n b^n c^n\}$
G:
$S \rightarrow aSBC$
$S \rightarrow e$
$CB \rightarrow BC$
$aB \rightarrow ab$
$bB \rightarrow bb$
$bC \rightarrow bc$
$cC \rightarrow cc$
Není těžké ověřit, že $L(G)=L=(\{a^n b^n c^n\})$.
G se dá převést na ekvivalentní kontextovou gramatiku G′:
pravidlo $CB \rightarrow BC$ se nahradí trojicí pravidel
$CB \rightarrow CB'$
$CB' \rightarrow BB'$
$BB' \rightarrow BC$.

7.2. Turingův stroj

Na úrovni nejvyšší tedy u jazyků typu 0 je akceptorem takového jazyka Turingův stroj. Budete se jím detailně zabývat v teorii vyčíslitelnosti a složitosti. Nyní si ho ukažme pouze jako ideu. V roce 1936 Alan Turing, který je pro teoretickou informatiku klíčovou postavou, formuloval svou ideu formalizace pojmu algoritmus ve formě Turingova stroje (TS). Tato formalizace má svůj velmi jednoduchý princip mechanismu se vstupní potenciálně nekonečnou páskou s danou abecedou a čtecí hlavou, která může **zapisovat i číst** na pásce a pohybovat se po jednom políčku. Schéma tohoto stroje lze vidět na obrázku. Lineárně omezený automat se liší jen

v tom, že páska pro něj není nekonečná, ale je omezena na k – násobek velikosti vstupního slova. Právě to pak způsobí, že není schopen rozpoznávat jazyky typu 0.

Turingův stroj



Tento velice jednoduchý formalismus s velkou výpočetní silou umožnil formulovat pro informatiku klíčové pojmy jako jsou rozhodnutelnost a částečná rozhodnutelnost problémů (příp. lze tyto pojmy aplikovat na funkce, množiny či jazyky). Podařilo se dokázat vlastnosti některých problémů (nejznámějším nerozhodnutelných problémem je problém zastavení). Myšlenky důkazů těchto faktů jsou poměrně jednoduché, i když netriviální a lze je najít v literatuře [Ja97a] a [Ch84]. Dalšími důležitými výsledky jsou vztahy mezi jazyky typu 0 a rekurzivně spočetnými jazyky, které spadají také do TFJA. Pro zájemce lze doporučit distanční studijní oporu pro tento kurz [Pa02].

Nejdůležitější probrané pojmy:

- Chomského hierarchie
- Generativní gramatika
- Turingův stroj

Úkol k textu:

Sestrojte ke každé třídě jazyků Chomského hierarchie pět jazyků, které do ní patří (s výjimkou typu 0).

8. Aplikace v programátorských úlohách

Cíl:

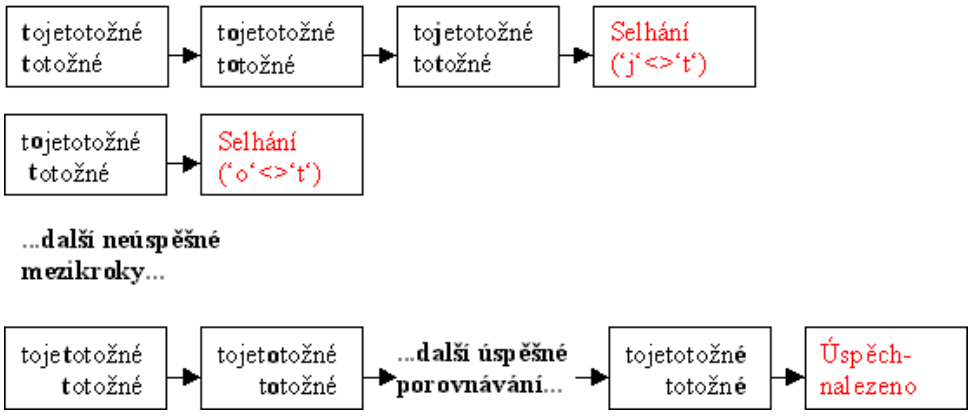
Po prostudování této kapitoly pochopíte:

- jednoduchý příklad, jak s pomocí KA vyložit netriviální algoritmus vyhledávání

8.1. Regulární jazyky a konečné automaty v praxi

Nejjednoduššími jazyky, které zkoumá teorie formálních jazyků, jsou jazyky regulární (resp. jazyky rozpoznatelné konečnými automaty). I když tyto pojmy zní odtažitě, podívejme se na jednu úlohu, kterou asi řešil každý čtenář tohoto článku a tou je vyhledávání v textu. Hned poté si ukážeme, jak možné takovou myšlenku ilustrativně vysvětlit pomocí pojmu konečného automatu. Pro tuto úlohu existují různě efektivní a složité algoritmy. Pokusme se rozebrat nejprve ten nejjednodušší, který nás zřejmě napadne (jde o algoritmus brute-force - brutální síla).

Intuitivní příklad: Vezměme si slovo „totožné“. Jak bychom realizovaly jeho vyhledávání například v textu „tojetotožné“.

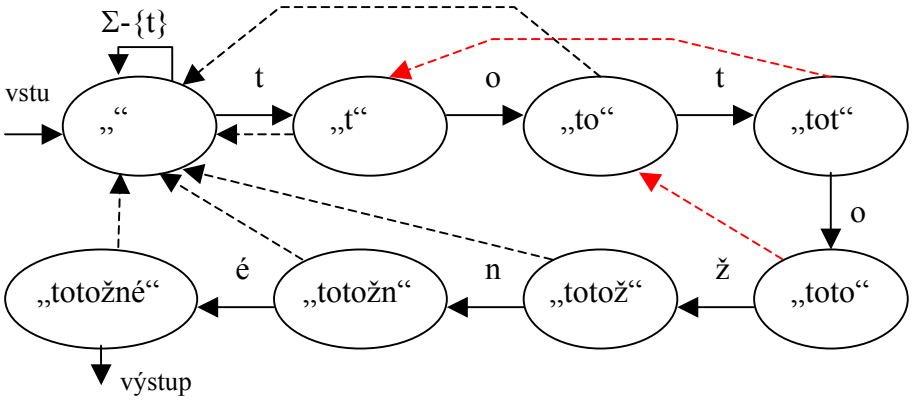


Algoritmy, které postupně načítají text a hledají výskyt slova, samozřejmě využívají knihoven funkcí. Tyto knihovny jistě obsahují již připravené algoritmy porovnání řetězců apod. Přesto půjdeme-li až na jádro způsobu nalezení slova bez použití těchto pomůcek, musí se číst postupně znaky textu a srovnávat – jde o první písmeno hledaného slova ‚t‘? Pokud ano, dále srovnávej zda souhlasí následující písmena... Pokud projdeme celé slovo totožný, aniž by se v právě načítaném úseku textu něco lišilo, pak můžeme skončit a říct, že „totožný“ se v textu vyskytuje a pokud ne, pak se vrátíme na další písmeno textu a celý postup opakujeme. Toto je zhruba

řečeno algoritmus brutální síly. Jeho postup probíhá zkráceně takto (srovnávání textu a slova):

Vidíte, že uvedený algoritmus je skutečně „brutální“. Nevyžaduje sice příliš mnoho uvažování, ale na druhou stranu je poměrně „hloupý“, protože se vždy vrací v textu na následující symbol a vůbec nevyužívá informaci o tom, co již v hledaném slově úspěšně srovnal. Proto by bylo rozumné zkusit navrhnout algoritmus, který by se již nemusel nikdy vracet v textu na symboly, které již srovnával. Pokusme se tuto myšlenku ilustrovat pomocí pojmů teorie formálních jazyků.

Z hlediska teorie formálních jazyků, jde o vyhledávání regulárního výrazu (mimochodem velice strukturálně jednoduchého – tyto výrazy mají obecně mnohem větší sílu než pro náš ilustrativní příklad), který můžeme realizovat pomocí konečného automatu. Pokusme se tento automat reprezentovat s pomocí stavového diagramu (jde o automat zobecněný s e-přechody – přerušovanými čarami, což nesnižuje obecnost):



řídící jednotka má kruhové symboly (stavy – určují zde, jaká část hledaného slova je již nalezena), které se během práce automatu mohou stávat aktivní (při začátku čtení slova po symbolech je aktivní stav do kterého vstupuje šipka), koncový stav je stav, ze kterého šipka vystupuje.
Šipky mezi stavy určují, jaký stav se stane aktivní místo stávajícího při přečtení symbolu, který je u šipky (nazývají se přechody mezi stavy).
Pozn. šipka ◄---- znamená, že automat přejde do určeného stavu v situaci, která není jinak určena (v tom případě nečte z textu žádný symbol); Σ-{t} označuje všechny přípustné symboly kromě ‚t‘.

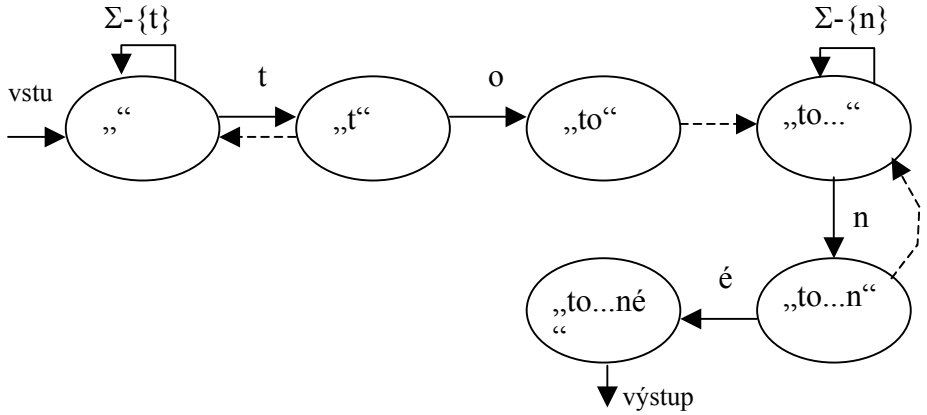
Co nám tento diagram říká? Stavy vyjadřují, jakou část hledaného slova jsme již úspěšně načetli. Na počátku po vstupu do automatu nejprve čekáme na symbol ‚t‘, kterým slovo začíná (to je ona smyčka Σ-{t}). Po načtení ‚t‘ se dostáváme do příslušného stavu. Pokud přijde ‚o‘ pokračujeme v úspěšném porovnávání, ale pokud ne, pak se musíme

rozhodnout, kam se vrátit, abychom sice nic z textu znovu zbytečně nečetli, ale zároveň abychom tak neopominuli již načtenou úspěšnou část slova. V tomto případě se můžeme vrátit až na počátek. Všechny situace s černou přerušovanou čarou jsou tyto „standardní návraty“. Podívejme se ale, co se děje, jsme-li již ve stavu „tot“. V tom případě se nemůžeme

vrátit úplně na počátek, protože bychom tak ignorovali, že jsme již úspěšně načetli symbol ‚t‘. Musíme se tedy do příslušného stavu nastavit, abychom tak neporušili potenciální možnost vyhledání slova v textu. Stejná „nestandardní“ situace nastává ve stavu „toto“. Musíme vzít v úvahu, že i přes neúspěch pro „totož“ jsme se již dostali do stavu, kdy je načteno „to“ a může potenciálně přijít „tot“! Naznačme schématicky, jak pracuje tento vylepšený algoritmus.

Tento automat nám vlastně poskytuje jakési „know-how“, díky němuž se zbavíme základního nedostatku (z hlediska efektivity algoritmu) a to je nutnost vracet se v textu vždy na další symbol. Při tomto postupu nikdy již čtený symbol v textu nemusíme znovu načítat. To jistě v rozsáhlých textech zrychlí hledání. Cena za to je nutnost zjistit si, kam se musím vrátit v automatu při selhání. Jelikož to však činíme jen jednou na počátku, u rozsáhlých textů se to vyplatí. Popsaný postup je vlastně teoreticky popsaným algoritmem známým jako Knuth-Morris-Prattův algoritmus. Lze jej naprogramovat a navíc poměrně jednoduše – je pouze třeba zkonstruovat postup vytváření „automatu“ – jinak řečeno tabulky, která popisuje kam se vrátit z jednotlivých stavů - jako algoritmus (není to složité – v podstatě jde o problém prohledávání slova v sobě samém a tím zjištění – kolik symbolů v jednotlivých částech slova se shoduje s jeho začátkem).

Příklad, který jsme pravě prezentovali, je samozřejmě velice jednoduchý. Konečné automaty mají větší možnosti než jen rozpoznávání pevných řetězců. Regulární výrazy, které k nim přísluší, mohou nahrazovat části slov libovolnými kombinacemi apod. Například lze sestrojit automat, který by vyhledával text se slovem, které začíná na „to“ a končí na „né“– tedy např. totožné, toporné, topné apod. Takový automat by vypadal následovně.



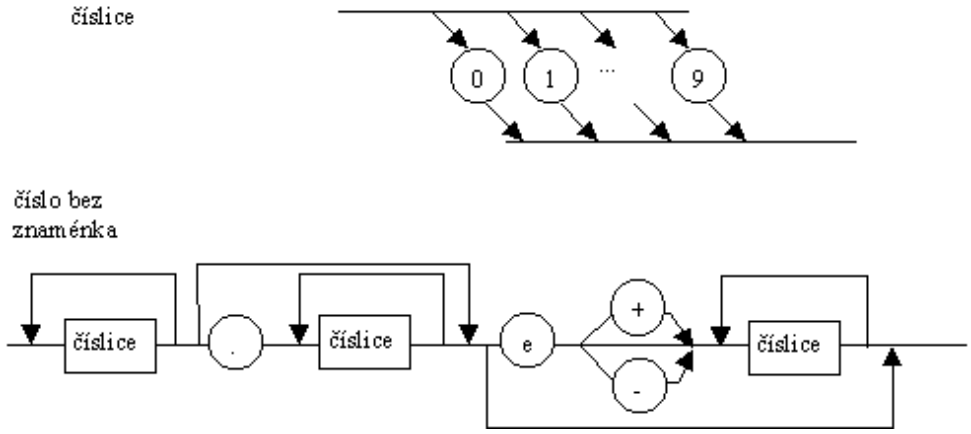
Automat nejprve hledá začátek „to“, pak může následovat cokoliv (jakákoliv kombinace symbolů). Přijde-li ‚n‘, přejde se do stavu to + cokoliv + ‚n‘. Nenásleduje-li ‚é‘, pak se nemusí jít na začátek, neboť „to...“ již máme vytvořeno.

Dalším typickým příkladem využití konečných automatů je vyhledávání několika různých slov v textu najednou nebo naopak, slov která splňují více podmínek najednou. Právě zde se dostáváme opět již k diskutované formalizaci. Pokud pochopíte, jak automat pracuje a jak je sestrojovat, můžete po důkladnějším studiu teorie formálních jazyků a automatů používat její formální aparát – tedy například algoritmy na sestrojování sjednocení, průniku automatů apod. Pokud Vás zaujal tento ilustrativní příklad, poznali jste, jak může být pro studenta zajímavé používat tento formalismus například v algoritmizaci při výuce vyhledávacích algoritmů. Věříme, že tato teorie přímo vybízí k hledání dalších zajímavých úloh, na kterých se mohou studenti seznámit s pochopitelnými a efektivními programátorskými technikami a zároveň tak poznat svět teoretické informatiky. Lze k tomu využít již zmiňované učebnice, ať již v elektronické nebo fyzické podobě. Také Internet je velkým zdrojem inspirace pro další didaktickou práci s konečnými automaty a regulárními výrazy. V další kapitole pouze naznačíme, k čemu směřuje další důležitá formalizace. Bylo by nad rámec tohoto omezeného článku ji rozebírat blíže, považujte ji proto za inspiraci, jak ve výuce nastínit studentům téma bezkontextových gramatik a syntaktické analýzy.

8.2. Bezkontextové gramatiky a syntaktická analýza

Třídou jazyků bezprostředně následující v hierarchii jsou bezkontextové jazyky. Jejich význam v praxi leží pro informatika především v oblasti umělých jazyků – programovacích, dotazovacích apod. Studenti, kteří se seznamují se základy algoritmizace a programují v konkrétním programovacím jazyce, by měli hned od začátku pracovat s jasně definovanými (syntaktickými) konstrukcemi jazyka. U následujícího příkladu je pro jednoduchost zvolena konstrukce na úrovni regulárních jazyků. Nicméně syntaktické diagramy lze využít především u struktur na úrovni bezkontextových jazyků.

Intuitivní příklad:
V programovacím jazyce Pascal se využívají čísla bez znaménka. Můžeme je považovat za syntaktickou strukturu, kterou lze velmi ilustrativně zobrazit následujícím způsobem.



Tato je pro studenty velmi vhodná a názorná, umožňuje sledovat v tomto konkrétním případě, že číslice je jedna z možných deseti a číslo se pak skládá ze sledu číslic (alespoň jedné), dále může následovat desetinná tečka a za ní opět sled číslic (ovšem tento element je nepovinný – může být přeskočen). Následuje (opět nepovinně) znak ‚e' (exponent), který se skládá ze sledu číslic, které mohou a nemusí být uvozeny znaménkem.

Tyto konstrukce mohou být zapsány pro celý jazyk Pascal v přesné notaci bezkontextové gramatiky (nebo v bohatší Backus-Nauerově formě viz skripta [Ce92]). K takovému zápisu je pak možné sestrojit LL(1) gramatiku, ze které lze názorným postupem sestrojit a implementovat analyzátor (algoritmus, který zjišťuje zda program v Pascalu neobsahuje chyby v syntaxi). To lze udělat na základě různých formálních metod teorie formálních jazyků a automatů. Nejilustrativnější z nich je pak metoda rekurzivního sestupu [Dv92],[Le02] nebo [Ka02], která umožňuje přímočaře ke gramatice napsat kód v libovolném strukturovaném programovacím jazyce.

V kvalitní učebnici jazyka Pascal [Ji88], která je i dnes hojně využívaná nejen pro výuku Pascalu, ale i principů algoritmizace a programování, je možné nalézt pomocí syntaktických diagramů i Backus-Nauerovy formy celou gramatiku jazyka. Pro studenta je velice důležité, aby si při používání programovacího jazyka uvědomoval, že program není jen posloupně zapsanou sekvencí příkazů, ale že program má svá pevná syntaktická pravidla, na kterých musí být sestaven.

Následující kód v jazyce Pascal ukazuje zpracovaný fragment analyzátoru pro typ číslo, podle syntaktických diagramů.

```
procedure Number;
var point : boolean;
begin
  point := false;
  while ch in numeric do
  begin
```

```
    ident := ident+ch; GetCharI;
    if ch in ['e', 'E', DecimalSeparator] then
    begin
      point := true;
      ident := ident+ch; GetCharI;
      if ch in ['+', '-'] then
      begin
        ident := ident+ch; GetCharI;
      end;
    end;
  end;
  if ch in ignore then GetChar;
  if point then
     AssignTerms(TReal.Create(ident), 0)
  else AssignTerms(TInteger.Create(ident), 0);
  ident := '';
end;
```

Procedura GetChar (resp. GetCharI) zabezpečuje načítání jednotlivých znaků z čísla. Vidíme, že procedura odpovídá zápisu syntaktického diagramu. Situaci, kdy se nějaký element jazyka podle syntaktického diagramu opakuje přesně simulujeme pomocí příkazu cyklu „while“ a podmíněný výskyt lze algoritmicky vyjádřit pomocí podmíněného příkazu „if“. V návaznosti na strukturu celého jazyka Pascal by bylo možné vytvořit systém navzájem se rekurzivně volajících procedur, které by zjišťovaly správnost kódu (syntaktický analyzátor) a generovaly výstupní formu (zde například funkce AssignTerms vytvářejí objekty reprezentující buď celé nebo reálné číslo). Uvedený fragment je vyňat z programu, který realizuje automatizovanou dedukci a používá syntaktický analyzátor pro formule predikátové logiky [Ha99].

Nejdůležitější probrané pojmy:

- algoritmy vyhledávání
- syntaktické diagramy

Úkol k textu:

Vyhledejte (např. na Internetu) popis algoritmu vyhledávání Aho-Corasickové a pokuste se interpretovat jeho funkci na příkladu pomocí konečného automatu.

9. Programátorské didaktické pomůcky

Cíl:

Po prostudování této kapitoly se naučíte:

- používat některé pomůcky, které Vám usnadní učení

V následující kapitole stručně popíšeme, které pomůcky můžete při učení používat. Půjde spíše o jejich výčet, neboť tyto pomůcky mají vlastní dokumentaci.

9.1. Počítačové aplikace jako programátorské didaktické pomůcky

Jak si ukážeme, tyto aplikace pro výuku teorie formálních jazyků podporují aspekty přispívající ke kvalitě výuky. Nejde jen o jejich využití jako interaktivních pomůcek pro pochopení teoretických předmětů, ale i o využití algoritmických dovedností, které fungují obousměrně. Přinášejí s sebou obousměrný transfer – studenti se naučí řešit teoretické postupy algoritmicky pomocí počítače a zároveň tak lépe pochopí teoretickou stránku problematiky. Vždyť podle psychologických výzkumů význam právě takovýchto pomůcek je neoddiskutovatelný ([Tu90] na základě psychologických studií Fredmanna):

Průměrný člověk si zapamatuje přibližně:
- 10% z toho, co přečte
- 20% z toho, co slyší
- 30% z toho, co vidí v podobě obrazu
- 50% z toho, co vidí a současně slyší
- 70% z toho, co vidí, slyší a aktivně vykonává
- 90% z toho, k čemu dospěl sám, na základě vlastních zkušeností vykonáváním činnosti

Programátorské didaktické pomůcky jsou prostředkem, který k vysoké efektivitě výuky směřuje. V následujících kapitolách se pokusíme některé z nich rozebrat, přičemž se zaměříme především na aplikaci GERDS, která byla samostatně vyvinuta autorem a nejlépe splňuje všechny atributy, které formulujeme pro takovou pomůcku. Dále rozebereme další aplikace, které však sice nenaplňují tak dobře všechny aspekty, ale jsou také pro výuku velmi efektivní a přínosné.

Za základní požadavky kladené na programátorskou didaktickou pomůcku lze považovat:
1. Jde o aplikaci realizovanou prostřednictvím (osobního) počítače (příp. i na jiném prostředku, který umožňuje jeho programování)
2. Tato aplikace řeší vybraný teoretický problém a umožňuje studentům interaktivně řešit problémové úkoly – tedy studenti mohou parametry programu i vstupu měnit podle vlastní vůle nebo pokynů učitele.

3. Aplikace má alespoň základní dokumentaci, která studentům umožní samostatnou práci a zároveň je vhodné, aby obsahovala ilustrativní příklady.
4. Procedury, které vybraný problém při zvolené konfiguraci řeší, je možné rozebrat na základě jejich algoritmické implementace. Zároveň lze do těchto procedur zasahovat a měnit chování programu, nebo použít částečné úseky kódu pro řešení příbuzných problémů.

Jak již bylo naznačeno, je splnění všech těchto podmínek obtížné. Splňují je v tomto textu beze zbytku pouze některé aplikace. Pokud jde o body 1. – 3. jsou podmínky jasné, ovšem naplnění bodu 4. naráží na mnoho komplikací:

- autorská práva k některým i akademickým aplikacím sice umožňují jejich použití, avšak omezují nebo zcela vylučují práci se zdrojovými kódy aplikace
- implementace je realizována v málo přístupném prostředku pro studenty (například jazyk C++ je sice z profesionálního hlediska účinný, ale jeho didaktické použití není příliš vhodné)
- aplikace používají velké množství již hotových knihoven nebo produktů, které zamezují vniknutí do jejich algoritmických řešení apod.

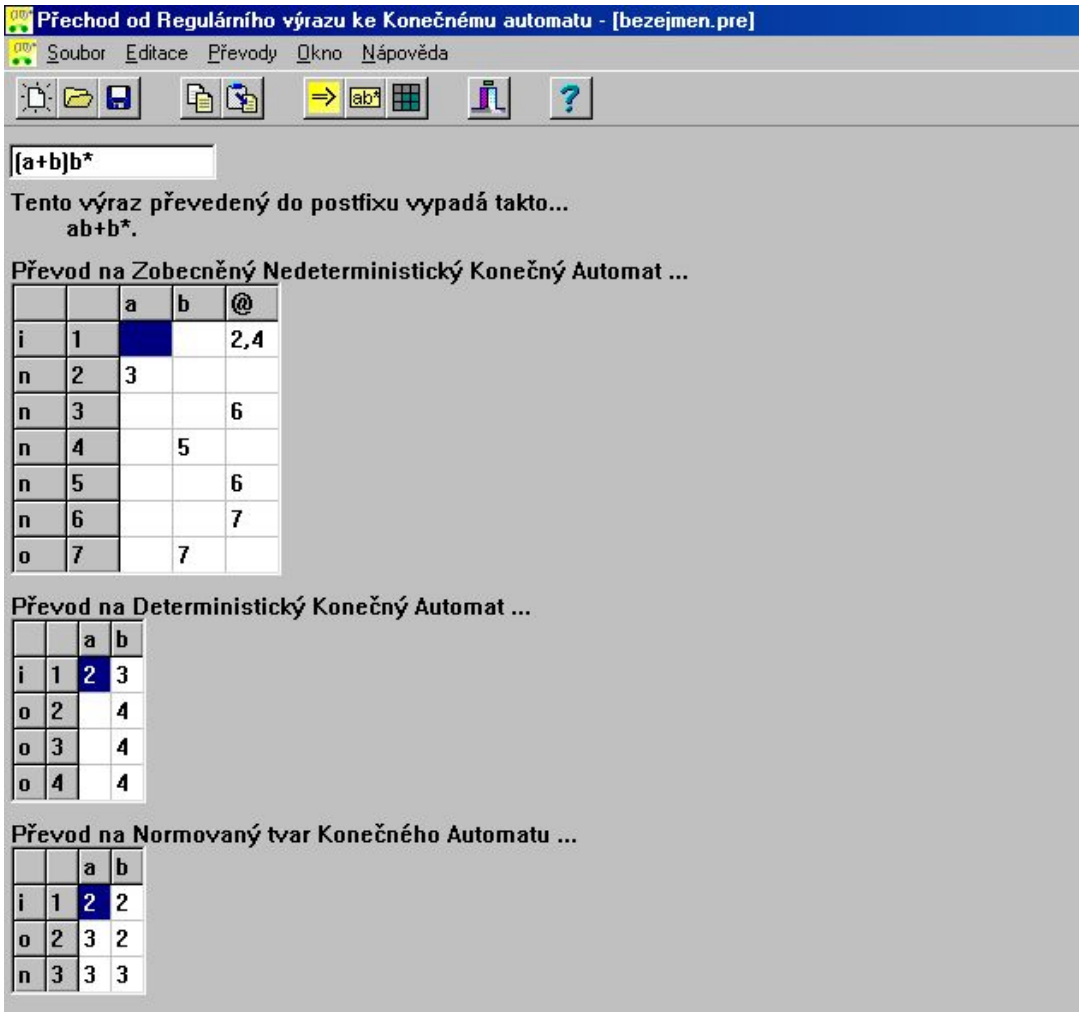
Vzhledem k tomu, že nelze tyto problémy pominout ani nelze zavrhnout aplikaci, která bod 4. nenaplňuje, doporučuje se splnění v přiměřených mezích. Například aplikace RABJ1 tento bod nebude splňovat, ale přesto je didaktický význam a řešení tak zdařilé, že lze jeho použití ve výuce teorie formálních jazyků velmi doporučit a nahradit teoretickým výkladem algoritmů, který tento prostředek implementuje.

9.2. PregJaut

Velmi užitečnou aplikací je PREGJAUT – bakalářská práce vytvořená na OU. Tato aplikace umožňuje provádět a zobrazovat detaily převodu regulárních výrazů na automaty (nedeterministické, deterministické, podílové a normované). Tento program studenti využívají pro kontrolu samostatně počítaných příkladů a také pro sledování průběhu převodu jednotlivými mezikroky. K programu jsou rovněž přiloženy zdrojové kódy, které mohou studenti použít pro lepší pochopení probíraných algoritmů. Aplikace je uložená v adresáři PregJaut a obsahuje poměrně solidně napsaný popis přechodu od teoretických postupů k jejich implementaci.

Algoritmy, jakožto jádro programu, jsou vytvořeny v Turbo Pascalu 7.0 a jsou uloženy ve třech unitech. První unit je UNITSYNT.PAS, ve kterém se provádí syntaktická analýza zadaného regulárního výrazu v infixu metodou s předsnímáním jednoho symbolu dopředu bez návratu s rozpoznáváním pěti druhů chyb v zápisu regulárního výrazu a s paralelním zápisem zadaného regulárního výrazu do postfixu. Rozpoznávané chyby jsou: chybí pravá závorka, chybí identifikátor nebo výraz, chybí operátor,

přebývá pravá závorka, špatný znak. Druhý unit je UNITMAIN.PAS, který převede regulární výraz zadaný v postfixu na ZNKA. Třetí unit je UNITPREV.PAS, který provádí všechny ostatní přechody (tzn. ZNKA -> NKA -> DKA -> Totální KA -> PA -> NA).
Ostatní unity jsou již vytvořeny programovacím nástrojem Delphi 2.0, který je použit pro grafickou nadstavbu programu. Hlavním datovým souborem je Delphi projekt PREGJAUT.DPR, který má v sobě informace o všech použitých unitech a formech. Hlavním formem je MainForm, jehož unit se nazývá UNITPRJA.PAS, který vyvolává a obhospodařuje ostatní formy a unity. Form, v kterém se vizualizují provedené algoritmy se nazývá MDIChild. Jeho vlastníkem je MainForm, ve kterém může být několik formů typu MDIChild. Unit patřící k formu MDIChild je CHILDWIN.PAS, ve kterém jsou veškeré kroky k vizualizaci provedených algoritmů, či různé jiné věci (jako např. zadání regulárního výrazu, tabulky, kontrola správnosti vyplnění tabulky, apod.). Další unity a k nim patřící okna jsou PREVFORM.PAS (jaké z převodů se mají vypsat na obrazovku), UNIT2.PAS (AboutBox), UNIT3.PAS (úprava vložené tabulky).
Pro práci se studenty se osvědčilo zadávat výpočet jednoduchého regulárního výrazu manuálně (např. ((a + b)b*)). Studenti si tak postup vyzkoušejí samostatně a uvidí, že tento postup lze efektivně realizovat automatizovaně. Dále je velmi užitečné jim ukázat, že ekvivalentní výrazy (např. (ab* + bb*)) vedou ke stejným automatům. Studenti jsou podobným aplikovaným způsobem výuky osloveni a sami si pak dokáží vytvářet problémové příklady.

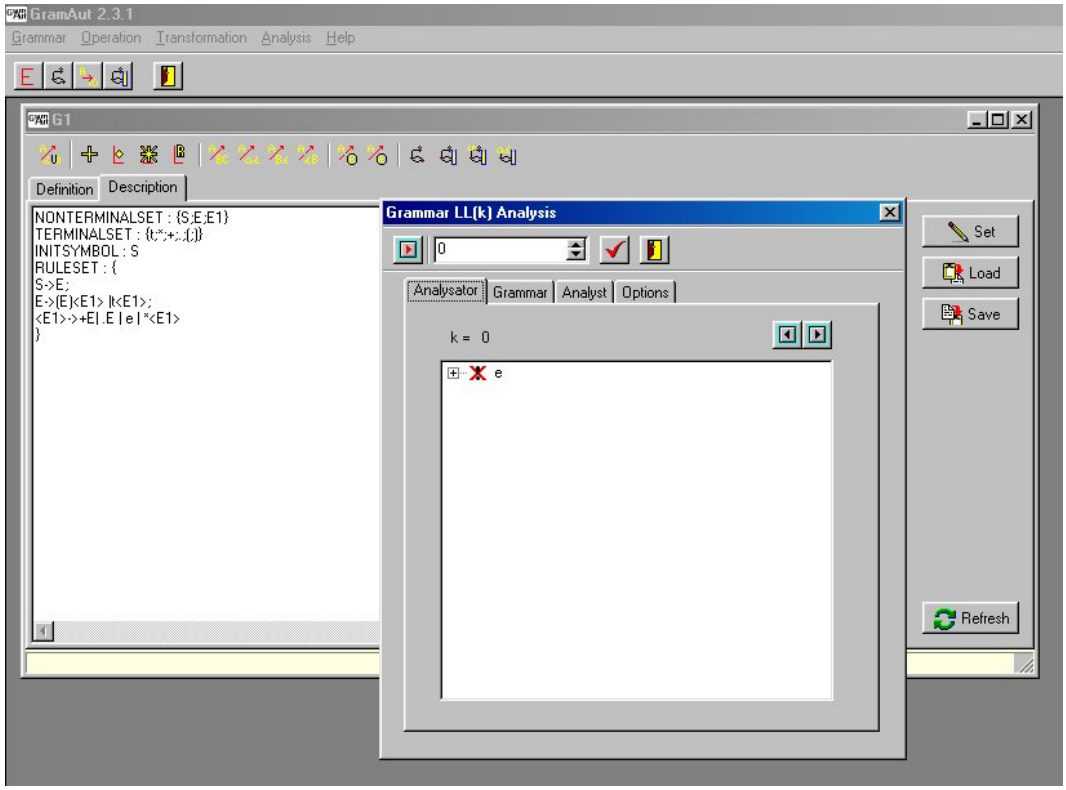


Uživatelské rozhraní PREGJAUTu je vidět na obrázku.

9.3. GramAut

Individuální tvůrčí práce studentů patří neoddělitelně k aplikačnímu pojetí výuky. Již dnes je realizována prostřednictvím seminárních, bakalářských a diplomových prací. Některé z nich nejenže prokazují hluboké pochopení učiva studenty, ale jsou dokonce zpětně využitelné ve výuce. Pokusíme se podívat alespoň na nejlepší z nich.

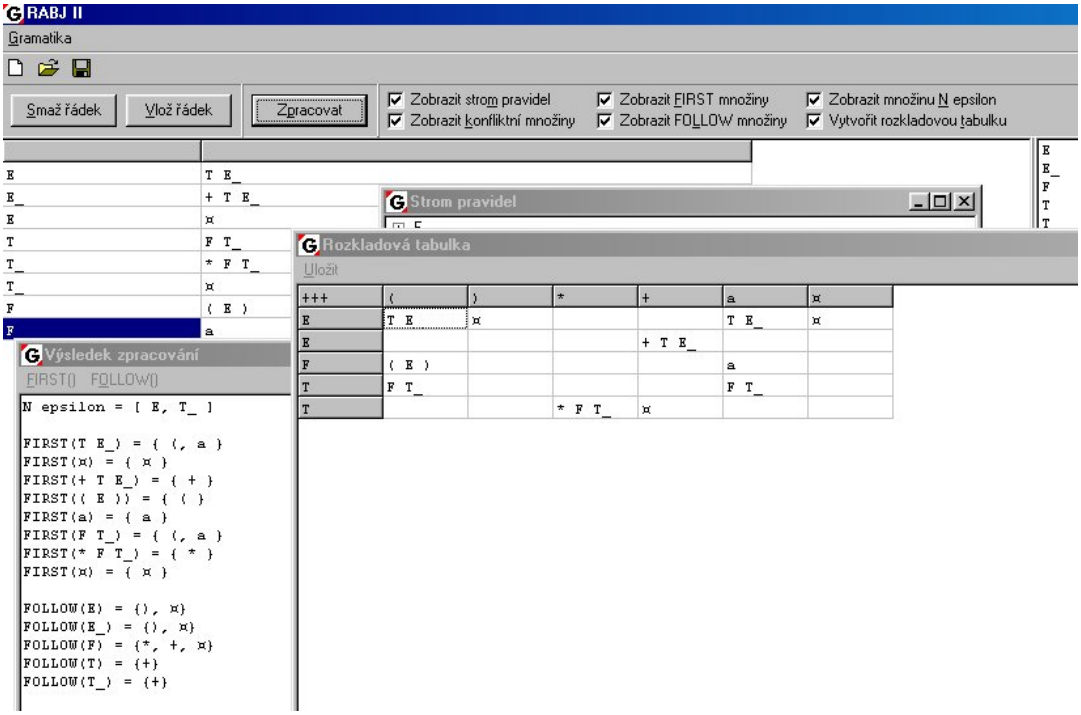
Asi nejrozsáhlejší a nejpreciznějí zpracovanou je práce Mgr. Hřivňáka [Hr02], která byla zpracována pod vedením Mgr. Pavlisky. Tato práce přináší počítačovou aplikaci GramAut, která umožňuje návrh a převody regulárních a bezkontextových gramatik a příslušejících automatů. Implementuje stávající algoritmy a dává velmi účinný didaktický nástroj pro výuku. Umožňuje i provádět syntaktickou analýzu u těch gramatik, které vyhovují podmínkám pro LL(k) resp. LR(k) gramatiku. Na obrázku



můžete vidět grafické rozhraní této aplikace, která implementuje netriviální algoritmy.

9.4. RABJ

Další velmi zdařilou prací (dokonce seminární!), která vznikla během výuky teorie formálních jazyků a automatů je aplikace studenta Koběrského pro výpočet množin FIRST a FOLLOW u LL(1) gramatik a vytvoření rozkladové tabulky. Všechny tyto práce přispívají nejen k osobnímu rozvoji studentů v programování, ale i v teoretické informatice, neboť jim umožňují lepší pochopení problematiky. Příklad, který můžete vidět na obrázku, je gramatikou pro aritmetické výrazy s operátory +,* a závorkami. Gramatiku lze navrhnout přímo v tomto programu a poté si lze nechat vypsat následující položky – strom pravidel, množiny FIRST, FOLLOW, N-epsilon, množinu konfliktů a vytvořit rozkladovou tabulku, pokud nenastal žádný konflikt.



Nejdůležitější probrané pojmy:

- programátorská pomůcka
- pomůcky pro výuku teorie formálních jazyků a automatů

Úkoly k textu:

1. Seznamte se s pomůckou PregJaut a připravte si několik příkladů regulárních výrazů, které jsou ekvivalentní a proveďte jejich převod na normované redukované automaty a ekvivalenci si tím dokažte.
2. Pomocí PregJautu si nasimulujte sjednocení dvou automatů prostřednictvím regulárního výrazu.
3. V pomůcce GramAut si vyzkoušejte probrané převody gramatik (redukovaná, nevypouštějící gramatika, zásobníkový automat atd..)

10. Vybrané partie teorie vyčíslitelnosti a složitosti

Cíl:

Tato kapitola Vám dává pokyny k prostudování vybraných témat z oblasti teorie vyčíslitelnosti a složitosti.

Jak bylo uvedeno v úvodním průvodci studiem, aby Vaše znalosti byly komplexní, budu po Vás požadovat, abyste prostudovali některé partie opory Vyčíslitelnost a složitost I. [Pa02]. Nepůjde o hloubku znalostí srovnatelnou s tou, která je požadována v této opoře. Spíše půjde o to, aby jste získali přehled o této teorii.

10.1. Vyčíslitelnost

Seznamte se s následujícími pojmy:
- problém
- rozhodnutelnost a částečná rozhodnutelnost
- rozhodnutelné množiny
- Turingův stroj
- nerozhodnutelné problémy

Vypracujte dva libovolné úkoly z této části opory [Pa02].

10.2. Složitost

Seznamte se s následujícími pojmy:
- časová a prostorová složitost (výpočet, třídy složitosti)
- NP-úplné problémy
- rozhodnutelné množiny
- Turingův stroj
- nerozhodnutelné problémy

Vypracujte dva libovolné úkoly z této části opory [Pa02].

Literatura

[Ce92] ČEŠKA, Milan, RÁBOVÁ, Zdena. Gramatiky a jazyky. Brno, VUT 1992.Dokument dostupný na URL: http://www.fit.vutbr.cz/study/courses/TI1/public/gj-1.3.pdf

[Dv92] DVOŘAK, Stanislav. Dekompozice a rekurzivní algoritmy. Grada 1992, Praha

[Ch84] CHYTIL, Milan. Automaty a gramatiky. Praha, SNTL 1984.

[Ha99] HABIBALLA, Hashim. PROBLEM SOLVING THROUGH FIRST-ORDER LOGIC (theory and practice of non-clausal resolution). Graduační práce na : Ostravská Univerzita, PřF. 1999. 59 s.

[Ho79] HOPCROFT, J.E., ULLMAN, J. D. Introduction to Automata theory, Languages and Computation. Addison-Wesley, Reading (Mass.), 1979

[Ka02] KASTENS, U.: Demonstration of parsing methods, http://www.uni-paderborn.de\fachbereich\AG\agkastens\compiler\parsdemo\index.html

[Le02] LEWIS, F.D. Recursive Descent Parsing, http://cs.engr.uky.edu\~lewis\essays\compilers\rec-des.html

[Ja97] JANČAR, Petr. Teorie jazyků a automatů. VŠB TU Ostrava, Dokument dostupný na URL: http://www.cs.vsb.cz/jancar/

[Ja97a] JANČAR, Petr. Vyčíslitelnost a složitost. VŠB TU Ostrava, Dokument dostupný na URL: http://www.cs.vsb.cz/jancar/

[Ji88] JINOCH, J., MULLER, K., VOGEL, J. Programování v jazyce Pascal. SNTL 1988, Praha

[Pa02] PAVLISKA, Viktor: Vyčíslitelnost a složitost I. distanční studijní text OU, 2002

[Tu90] TUREK, Ivan: Didaktika technických predmetov, SNTL 1990, Bratislava

Na Internetu lze najít množství odkazů a materiálů – především v angličtině, nicméně existují i české a slovenské webovské stránky s materiály: např. http://www.fimuni.org/ apod. (leden 2003)