

Chomského normálny tvar, CYK algoritmus; Množiny *FIRST, FOLLOW*

Ing. Viliam Hromada, PhD.

C-510
Ústav informatiky a matematiky
FEI STU

`viliam.hromada@stuba.sk`



Chomského normálny tvar (forma) gramatiky

Definícia

Bezkontextová gramatika $G = (N, T, P, S)$ je v **Chomského normálnom tvare**, ak každé pravidlo je v jednom z tvarov:

1. $A \rightarrow BC$ pre $A \in N; B, C \in (N - \{S\})$;
2. $A \rightarrow a$ pre $A \in N, a \in T$;
3. $S \rightarrow \varepsilon$.



Úprava do CHNT

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$

Výstup: Gramatika G v CHNT

- 1: Odstráň z gramatiky ε -pravidlá.
- 2: Uprav gramatiku, aby počiatočný neterminál nebol v pravej strane žiadneho pravidla.
- 3: Odstráň z gramatiky jednoduché pravidlá.
- 4: Odstráň z gramatiky nadbytočné a nedostupné symboly.
- 5: **pre všetky** pravidlá tvaru $A \rightarrow X_1 X_2 \dots X_n, n \geq 3$ **rob**
- 6: nahrad' pravidlami $A \rightarrow X_1 A_1, A_1 \rightarrow X_2 A_2, \dots, A_{n-2} \rightarrow X_{n-1} X_n$, kde $A_1, \dots, A_{n-2}$ sú nové neterminály.
- 7: **koniec pre**
- 8: **pre všetky** pravidlá tvaru $A \rightarrow Y_1 Y_2$, kde $Y_1 \in T \vee Y_2 \in T$ **rob**
- 9: nahrad' na pravej strane pravidiel terminály $Y_1 (Y_2)$ novými neterminálmi $V_1 (V_2)$ a pridaj pravidlo (pravidlá) $V_1 \rightarrow Y_1$, resp. $V_2 \rightarrow Y_2$
- 10: **koniec pre**



Krok č. 2 z algoritmu

V algoritme sa po odstránení ε -pravidiel robí úprava: *Uprav gramatiku, aby počiatočný neterminál nebol v pravej strane žiadneho pravidla..*

- V prípade, že sa pri odstránení ε pridal do gramatiky nový neterminál $\acute{S}$ a pravidlá $\acute{S} \rightarrow \varepsilon \mid S$, tak tento krok je irelevantný.
- V prípade, že sa v gramatike jej **aktuálny** počiatočný neterminál S nachádza v niektorej pravej strane niektorého pravidla, **pridá sa nový počiatočný neterminál** $\acute{S}$ a pravidlo $\acute{S} \rightarrow S$.



CHNT - príklad

Gramatika $G = (\{S, A\}, \{a, b, c\}, P, S)$ s pravidlami:

$$S \rightarrow aASa \mid c$$

$$A \rightarrow Abc \mid c$$

- 1) ε -pravidlá nie je potrebné odstrániť - žiadne neobsahuje.
- 2) Zavedieme symbol $\acute{S}$ a pridáme pravidlo:

$$\acute{S} \rightarrow S$$

$$S \rightarrow aASa \mid c$$

$$A \rightarrow Abc \mid c$$



CHNT - príklad (pokr.)

3,4) Po odstránení jednoduchých pravidiel a nadbytočných symbolov:

$$\acute{S} \rightarrow aASa \mid c$$

$$S \rightarrow aASa \mid c$$

$$A \rightarrow Abc \mid c$$

5-7) Úprava pravidiel dlhších ako 3 symboly:

$$\acute{S} \rightarrow aA_1 \mid c$$

$$S \rightarrow aA_1 \mid c$$

$$A \rightarrow AA_3 \mid c$$

$$A_1 \rightarrow AA_2$$

$$A_2 \rightarrow Sa$$

$$A_3 \rightarrow bc$$



CHNT - príklad (pokr.)

8-10) Odstránenie terminálov z pravidiel s pravou stranou dĺžky 2

$$\acute{S} \rightarrow V_1 A_1 \mid c$$

$$S \rightarrow V_1 A_1 \mid c$$

$$A \rightarrow AA_3 \mid c$$

$$A_1 \rightarrow AA_2$$

$$A_2 \rightarrow SV_1$$

$$A_3 \rightarrow V_2 V_3$$

$$V_1 \rightarrow a$$

$$V_2 \rightarrow b$$

$$V_3 \rightarrow c$$



CHNT - poznámky

- Ak vieme v danej gramatike odvodiť slovo w , potom dĺžka jeho odvodu v CHNT je $2|w| - 1$.
- Strom odvodu (derivačný strom) slova v gramatike v CHNT je binárny strom.



Praktické využitie Chomského normálneho tvaru

- Chomského normálny tvar sa v praxi využíva v tzv. CYK algoritme.
- CYK algoritmus je **polynomiálnym** algoritmom na zistenie, či **ľubovoľná bezkontextová gramatika** $G = (N, T, P, S)$ generuje zadaný **reťazec terminálov** $w \in T^*$, teda či $w \in L(G)$.



CYK algoritmus

- Predpokladajme, že máme danú bezkontextovú gramatiku $G = (N, T, P, S)$ v **Chomského normálnom tvare**.
- Chceme zistiť, či slovo $w \in L(G)$, kde $w = a_1 \dots a_n$, $a_i \in T$, teda $|w| = n$.
- Ak $w = \varepsilon$, stačí sa pozrieť, či je v gramatike pravidlo $S \rightarrow \varepsilon$.
- Pre $w \in T^+$ vie CYK algoritmus (Cocke-Younger-Kasami) zistiť či $w \in L(G)$ so zložitou $O(n^3 \cdot |P|)$, kde n je dĺžka analyzovaného slova pre **hocijakú** bezkontextovú gramatiku a $|P|$ počet pravidiel gramatiky.
- Základnou ideou algoritmu je konštrukcia množín:

$$N_{i,j} = \{A \mid A \Rightarrow^* a_i a_{i+1} \dots a_j, A \in N\},$$

$$1 \leq i \leq j \leq n.$$

- Ak $S \in N_{1,n}$ potom slovo $w \in L(G)$.



CYK algoritmus

- Postupne sa hľadajú neterminály, ktoré vedia generovať podreťazce daného slova w .
- Na začiatku sa určia neterminály generujúce podreťazce dĺžky 1.
 - Keďže gramatika je v CHNT, dané neterminály sa určia priamo z pravidiel.
- Následne sa využíva nasledovná vlastnosť:
 1. Ak $B \in N_{j,k}$, t.j. $B \Rightarrow^* a_j \dots a_k$,
 2. ak $C \in N_{k+1,j+i}$, t.j. $C \Rightarrow^* a_{k+1} \dots a_{j+i}$,
 3. a zároveň existuje pravidlo $A \rightarrow BC$, potom
 4. $A \in N_{j,j+i}$, pretože $A \Rightarrow BC \Rightarrow^* a_j \dots a_k a_{k+1} \dots a_{j+i}$.



CYK algoritmus I

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$ v CNF, slovo $w \neq \varepsilon$

Výstup: Zistenie, či $w \in L(G)$

- 1: $w = a_1 \dots a_n$
- 2: **pre všetky** $i \in \{1, \dots, n\}$ **rob**
- 3: **pre všetky** $j \in \{i, \dots, n\}$ **rob**
- 4: $N_{i,j} \leftarrow \emptyset$
- 5: **koniec pre**
- 6: **koniec pre**
- 7: **pre všetky** $i \in \{1, \dots, n\}$ **rob**
- 8: **pre všetky** pravidlá tvaru $A \rightarrow a_i$ **rob**
- 9: $N_{i,i} \leftarrow N_{i,i} \cup \{A\}$
- 10: **koniec pre**
- 11: **koniec pre**
- 12: **pre všetky** $i \in \{1, 2, \dots, n-1\}$ **rob**
- 13: **pre všetky** $j \in \{1, 2, \dots, n-i\}$ **rob**



CYK algoritmus II

```
14:   pre všetky  $k \in \{j, \dots, j + i - 1\}$  rob
15:     pre všetky pravidlá  $A \rightarrow BC$ , kde  $B \in N_{j,k}$  a zároveň  $C \in N_{k+1,j+i}$  rob
16:        $N_{j,j+i} \leftarrow N_{j,j+i} \cup \{A\}$ 
17:     koniec pre
18:   koniec pre
19: koniec pre
20: koniec pre
21: ak  $S \in N_{1,n}$  potom
22:   vráť „Slovo patrí do jazyka. “
23: inak
24:   vráť „Slovo nepatrí do jazyka. “
25: koniec ak
```



CYK algoritmus - príklad

Je daná gramatika:

$$S \rightarrow AB \mid AC \mid a$$

$$A \rightarrow DA \mid a$$

$$B \rightarrow b \mid d$$

$$C \rightarrow BC \mid c$$

$$D \rightarrow d$$

Zistite pomocou CYK algoritmu, či slovo $dab \in L(G)$.



CYK algoritmus - príklad (pokr.)

Najprv sa zostroja množiny

$N_{1,1} = \{B, D\}$, kvôli pravidlám $B \rightarrow d$ a $D \rightarrow d$.

$N_{2,2} = \{S, A\}$, kvôli pravidlám $S \rightarrow a$ a $A \rightarrow a$.

$N_{3,3} = \{B\}$, kvôli pravidlu $B \rightarrow b$.

Ďalej podľa iterácii:

$i = 1, j = 1, k = 1$ $A \rightarrow DA, D \in N_{1,1}, A \in N_{2,2}$
teda $A \in N_{1,2}$

$i = 1, j = 2, k = 2$ $S \rightarrow AB, A \in N_{2,2}, B \in N_{3,3}$
teda $S \in N_{2,3}$

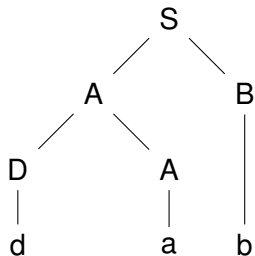
$i = 2, j = 1, k = 1$ $N_{1,1} = \{B, D\}, N_{2,3} = \{S\}$
do $N_{1,3}$ nepribudne nič

$i = 2, j = 1, k = 2$ $S \rightarrow AB, A \in N_{1,2}, B \in N_{3,3}$
teda $S \in N_{1,3}$

Keďže $S \in N_{1,3}$, tak $dab \in L(G)$.



Na základe príslušných množín vieme teoreticky aj nájsť derivačný strom, resp. odvodenie:



- Základnou úlohou CYK je zistiť, či slovo patrí do jazyka generovaného gramatikou, avšak v podstate sa dá použiť aj na zostrojenie derivačného stromu (na základe množín $N_{i,j}$).
- Výhodou CYK algoritmu je, že vie spracovať všetky bezkontextové gramatiky.
 - Aj nejednoznačné, aj jednoznačné
 - Aj nedeterministické, aj deterministické
- Nevýhodou CYK algoritmu je, že je potrebné, aby gramatika bola v CHNT a taktiež má kubickú zložitosť vzhľadom na dĺžku vstupného slova.
- Existujú aj ďalšie algoritmy, ktoré síce nevedia spracovať všetky bezkontextové gramatiky, avšak majú lineárnu zložitosť vzhľadom na dĺžku vstupného slova - viac si povieme pri syntaktickej analýze.



Vetná forma

Nech $G = (N, T, P, S)$ je formálna gramatika. Potom ľubovoľný reťazec $\alpha \in (N \cup T)^*$ taký, že

$$S \Rightarrow^* \alpha$$

nazývame **vetná forma** gramatiky G .

Vetná forma je teda ľubovoľný reťazec nad abecedou $N \cup T$, teda pozostávajúci z terminálov a neterminálov gramatiky, ktorý je možné odvodiť z počiatočného neterminálu S .

Jazyk generovaný gramatikou $L(G)$ je teda množina tých vetných foriem, ktoré zároveň predstavujú reťazce nad terminálmi gramatiky.



Analýza vlastností bezkontextových gramatík

- Je zrejmé, že to, aké reťazce generujú bezkontextové gramatiky - a či existuje derivácia nejakého reťazca v gramatike - úzko súvisí s pravidlami danej gramatiky.
- Preto sa pri bezkontextových gramatikách vyšetrujú tzv. množiny *FIRST* a *FOLLOW*, ktoré bližšie špecifikujú, aké vlastnosti majú vetné formy, resp. reťazce vznikajúce počas derivácií.
- Následne má potom zmysel tieto vlastnosti využívať pri zisťovaní, či derivácie reťazcov v gramatike existujú alebo nie.
- Množina *FIRST* pojednáva o tom, aké terminálne symboly môžu stáť **na prvom mieste** reťazca, ktorý viem derivovať z nejakej postupnosti symbolov gramatiky.
- Množina *FOLLOW* pojednáva o tom, aké terminálne symboly môžu **nasledovať** nejaký neterminál vo vetrnej forme.



Množina *FIRST*

Definícia

Nech $G = (N, T, P, S)$ je redukovaná bezkontextová gramatika a $\alpha \in (N \cup T)^*$.
Potom

$$FIRST(\alpha) = \{a \in T \mid \alpha \Rightarrow^* a\beta, \beta \in (N \cup T)^*\} \cup \{\varepsilon \mid \alpha \Rightarrow^* \varepsilon\}. \quad (1)$$

T.j. pre reťazec zložený z neterminálov a terminálov α je $FIRST(\alpha)$ množina terminálov, ktorými môžu začínať reťazce odvodené z α . A ak je možné z α odvodiť ε , tak do $FIRST(\alpha)$ patrí aj ε .



Množina *FIRST* - rôzne situácie

- Definícia hovorí, že množinu *FIRST* môžeme hľadať pre ľubovoľný reťazec nad symbolmi gramatiky.
- V praxi sa najprv nájde množina *FIRST* pre jednotlivé neterminály gramatiky, t.j. *FIRST*(*A*) pre všetky neterminály $A \in N$.
- Následne sa pri hľadaní množiny *FIRST*(α) pre ľubovoľné reťazce $\alpha \in (N \cup T)^*$ vychádza z jednotlivých množín *FIRST* pre neterminály.



Množina *FIRST* - rôzne situácie

1. ak gramatika obsahuje pravidlo $A \rightarrow a\alpha$, potom $a \in FIRST(A)$
2. ak $A \in N_\epsilon$, potom $\epsilon \in FIRST(A)$
3. ak gramatika obsahuje pravidlo $A \rightarrow \alpha$, potom $FIRST(\alpha) \subseteq FIRST(A)$



Množina *FIRST* - příklad

Nech je daná gramatika $G = (N, T, P, S)$, kde
 $N = \{S, A, B, C\}$, $T = \{a, b, +, (,), \$\}$ a pravidlá:

$$S \rightarrow C\$$$

$$A \rightarrow b \mid \varepsilon$$

$$B \rightarrow +S \mid \varepsilon$$

$$C \rightarrow A(C) \mid aB$$

Potom:

- $FIRST(B) = \{+, \varepsilon\}$
- $FIRST(C) = \{b, a, (\}$
- $FIRST(A) = \{b, \varepsilon\}$
- $FIRST(S) = \{a, b, (\}$

$$FIRST(aBA) = \{a\}$$

$$FIRST(AB) = \{b, \varepsilon, +\}$$

$$FIRST(A\$B) = \{b, \$\}$$



Množina $FIRST(X)$ I

Vstup: Redukovaná bezkontextová gramatika $G = (N, T, P, S)$.

Výstup: Množiny $FIRST(X)$ pre $X \in N \cup T$

- 1: **pre všetky** $A \in N$ **rob**
- 2: $FIRST(A) \leftarrow \emptyset$
- 3: **koniec pre**
- 4: **pre všetky** $a \in T$ **rob**
- 5: $FIRST(a) \leftarrow \{a\}$
- 6: **koniec pre**
- 7: **pre všetky** pravidlá $A \rightarrow a\alpha$ **rob**
- 8: $FIRST(A) \leftarrow FIRST(A) \cup \{a\}$
- 9: **koniec pre**
- 10: **pre všetky** $A \in N_\epsilon$ **rob**
- 11: $FIRST(A) \leftarrow FIRST(A) \cup \{\epsilon\}$
- 12: **koniec pre**



Množina $FIRST(X)$ II

- 13: **opakuj**
- 14: **pre všetky** pravidlá $A \rightarrow B_\alpha$ **rob**
- 15: $FIRST(A) \leftarrow FIRST(A) \cup FIRST(B_\alpha)$
- 16: **koniec pre**
- 17: **pokiaľ** zmenila sa niektorá z množín $FIRST(A), A \in N$



Množina $FIRST(\alpha)$

Vstup: množiny $FIRST(X)$, $X \in N \cup T$; reťazec $\alpha \in (N \cup T)^*$.

Výstup: $FIRST(\alpha)$, $\alpha \in (N \cup T)^*$

- 1: **ak** $\alpha = \varepsilon$ **potom**
- 2: **vráť** $\{\varepsilon\}$
- 3: **inak**
- 4: nech $\alpha = X_1 X_2 \dots X_n$
- 5: $FIRST(\alpha) \leftarrow FIRST(X_1) - \{\varepsilon\}$
- 6: $i \leftarrow 1$
- 7: **pokiaľ** $\varepsilon \in FIRST(X_i) \wedge i < n$ **rob**
- 8: $i \leftarrow i + 1$
- 9: $FIRST(\alpha) \leftarrow FIRST(\alpha) \cup (FIRST(X_i) - \{\varepsilon\})$
- 10: **koniec pokiaľ**
- 11: **ak** $i = n \wedge \varepsilon \in FIRST(X_n)$ **potom**
- 12: $FIRST(\alpha) \leftarrow FIRST(\alpha) \cup \{\varepsilon\}$
- 13: **koniec ak**
- 14: **vráť** $FIRST(\alpha)$
- 15: **koniec ak**



Množina *FIRST* - příklad

Nech je daná gramatika $G = (N, T, P, S)$, kde
 $N = \{S, A, B, C\}$, $T = \{a, b, +, (,), \$\}$ a pravidlá:

$$S \rightarrow C\$$$

$$A \rightarrow b \mid \varepsilon$$

$$B \rightarrow +S \mid \varepsilon$$

$$C \rightarrow A(C) \mid aB$$

Určte $FIRST(X)$ pro všechny $X \in N \cup T$, $FIRST(BS)$, $FIRST(AB)$, $FIRST(CB)$



Množina *FIRST* - príklad

FIRST(X):

<i>FIRST</i>	S	A	B	C	a	b	$+$	$($	$)$	$\$$
Terminály:					a	b	$+$	$($	$)$	$\$$
Pravidlá $A \rightarrow a\alpha$		b	$+$	a						
Neterminály $A \in N_\epsilon$		ϵ	ϵ							
1. iterácia:	a			$b, ($						
2. iterácia:	$b, ($									
3. iterácia:										
Celkom:	$a, b, ($	b, ϵ	$+, \epsilon$	$a, b, ($	a	b	$+$	$($	$)$	$\$$

Množina *FIRST* - příklad

$$FIRST(BS) = \{+, a, b, (\}$$

$$FIRST(AB) = \{b, +, \epsilon\}$$

$$FIRST(CB) = \{a, b, (\}$$



Množina *FOLLOW*

Definícia

Nech $G = (N, T, P, S)$ je redukovaná bezkontextová gramatika a $A \in N$. Potom

$$FOLLOW(A) = \{a \in T \mid S \Rightarrow^+ \alpha A a \beta\} \cup \{\varepsilon \mid S \Rightarrow^* \gamma A\}, \quad (2)$$

kde $\alpha, \beta, \gamma \in (N \cup T)^*$.

T.j. pre neterminál A je množina $FOLLOW(A)$ množina takých terminálov, ktoré môžu v nejakej vetnej forme stáť **hneď za neterminálom** A . Do množiny patrí aj ε , ak môže neterminál A stáť **na konci** nejakej vetnej formy.



Množina *FOLLOW*(A)

- Predpokladajme, že gramatika obsahuje pravidlo
 $A \rightarrow \alpha B \beta; \alpha, \beta \in (N \cup T)^*; A, B \in N$
- Predpokladajme, že existuje odvodenie: $S \Rightarrow^* \dots A a \dots \Rightarrow^* \dots \alpha B \beta a \dots \Rightarrow^* \dots$
 1. Vždy platí, že $\varepsilon \in FOLLOW(S)$
 2. $a \in FOLLOW(A)$
 3. $(FIRST(\beta) - \{\varepsilon\}) \subseteq FOLLOW(B)$
 4. (ak $\varepsilon \in FIRST(\beta)$), potom $FOLLOW(A) \subseteq FOLLOW(B)$.



Príklad: Nech je daná gramatika $G = (N, T, P, S)$, kde $N = \{S, A, B, C\}$, $T = \{a, b, +, (,), \$\}$ a pravidlá:

$$S \rightarrow C\$$$

$$A \rightarrow b \mid \varepsilon$$

$$B \rightarrow +S \mid \varepsilon$$

$$C \rightarrow A(C) \mid aB$$

Potom:

$$FOLLOW(S) = \{\varepsilon, \$,)\}$$

$$FOLLOW(A) = \{(}$$

$$FOLLOW(B) = \{\$,)\}$$

$$FOLLOW(C) = \{\$,)\}$$



Množina $FOLLOW(A)$ pre $A \in N$

Vstup: Redukovaná bezkontextová gramatika $G = (N, T, P, S)$, množiny $FIRST(X)$ pre $X \in N \cup T$

Výstup: množiny $FOLLOW(A)$ pre $A \in N$

- 1: **pre všetky** $A \in N$ **rob**
- 2: $FOLLOW(A) \leftarrow \emptyset$
- 3: **koniec pre**
- 4: $FOLLOW(S) \leftarrow \{\epsilon\}$
- 5: **opakuj**
- 6: **pre všetky** $B \in N$ v pravých stranách pravidiel $A \rightarrow \alpha B \beta$ **rob**
- 7: $FOLLOW(B) \leftarrow FOLLOW(B) \cup (FIRST(\beta) - \{\epsilon\})$
- 8: **ak** $\epsilon \in FIRST(\beta)$ **potom**
- 9: $FOLLOW(B) \leftarrow FOLLOW(B) \cup FOLLOW(A)$
- 10: **koniec ak**
- 11: **koniec pre**
- 12: **pokiaľ** sa zmenila niektorá z množín $FOLLOW(B)$, $B \in N$



Množina *FOLLOW*(*A*) - príklad

Príklad: Nech je daná gramatika $G = (N, T, P, S)$, kde
 $N = \{S, A, B, C\}$, $T = \{a, b, +, (,), \$\}$ a pravidlá:

$$S \rightarrow C\$$$

$$A \rightarrow b \mid \varepsilon$$

$$B \rightarrow +S \mid \varepsilon$$

$$C \rightarrow A(C) \mid aB$$



Množina *FOLLOW*(A) - príklad (pokr.)

<i>FOLLOW</i>	<i>S</i>	<i>A</i>	<i>B</i>	<i>C</i>
Začiatočný symbol	ε			
1. iterácia: $S \rightarrow \underline{C}\$$				\$
$B \rightarrow +\underline{S}$				
$C \rightarrow \underline{A}(C)$		(		
$C \rightarrow A(\underline{C})$				)
$C \rightarrow a\underline{B}$			\$(,)	
2. iterácia: $S \rightarrow \underline{C}\$$				
$B \rightarrow +\underline{S}$	\$(,)			
...				

Zvyšok tabuľky už budú prázdne riadky, pričom po 3. iterácii sa už množiny nezmenia a algoritmus končí. **V tabuľke uvádzame len novo-zistené symboly.**



Použitá literatúra

Aho, A., Lam, M., Sethi, R., Ullman, J.: *Compilers: Principles, techniques and tools*.

Dedera, L': *Počítačové jazyky a ich spracovanie*.

Linz, P.: *An Introduction to Formal Languages and Automata*.

