Plán přednášky

Výpočetní modely pro rozpoznávání bezkontextových jazyků

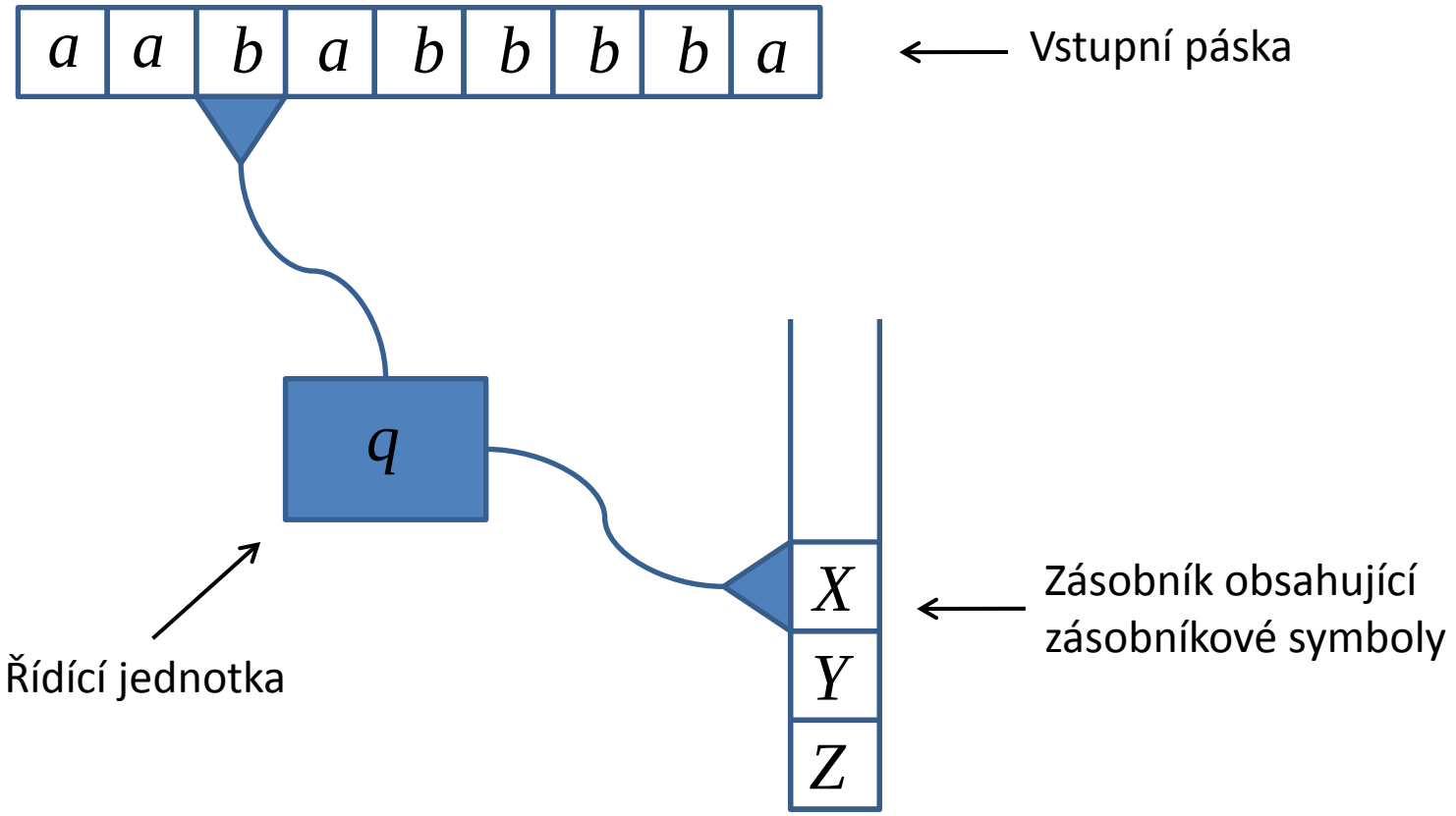
- zásobníkové automaty
- LL(k) a LR(k) analyzátory

Obecný algoritmus pro parsování bezkontextových jazyků

- dynamické programování

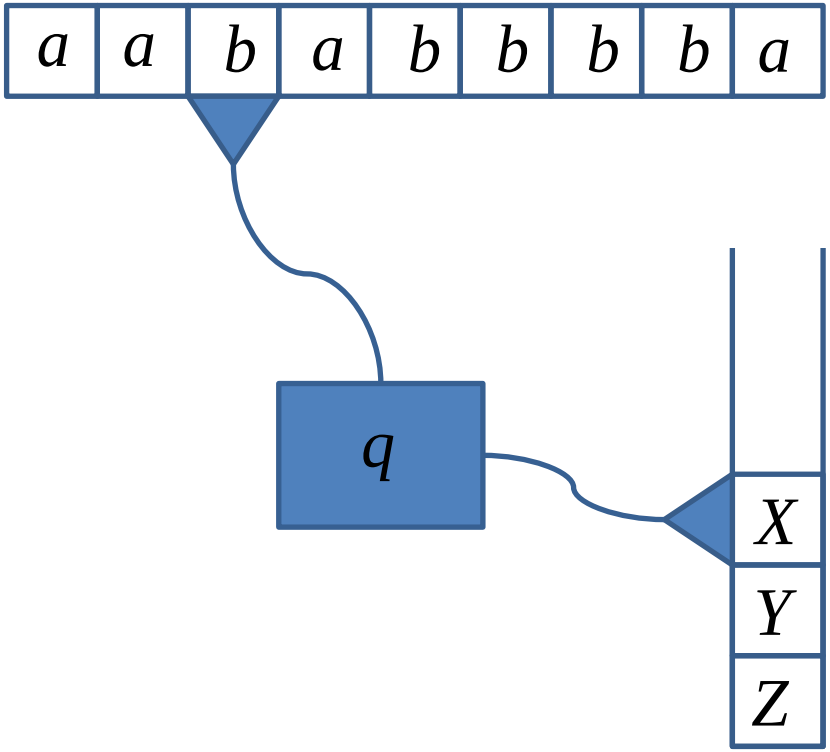
Zásobníkový automat

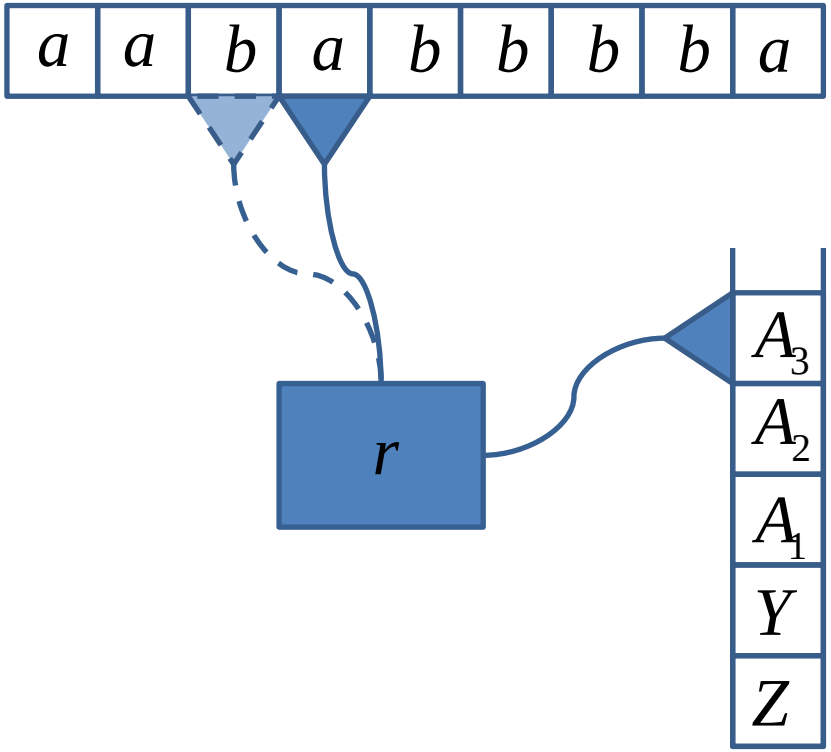
Schéma, stav v průběhu výpočtu:



Zásobníkový automat – výpočetní krok

Dvě možnosti: s posunem hlavy nebo bez posunu.





Zásobníkový automat – definice

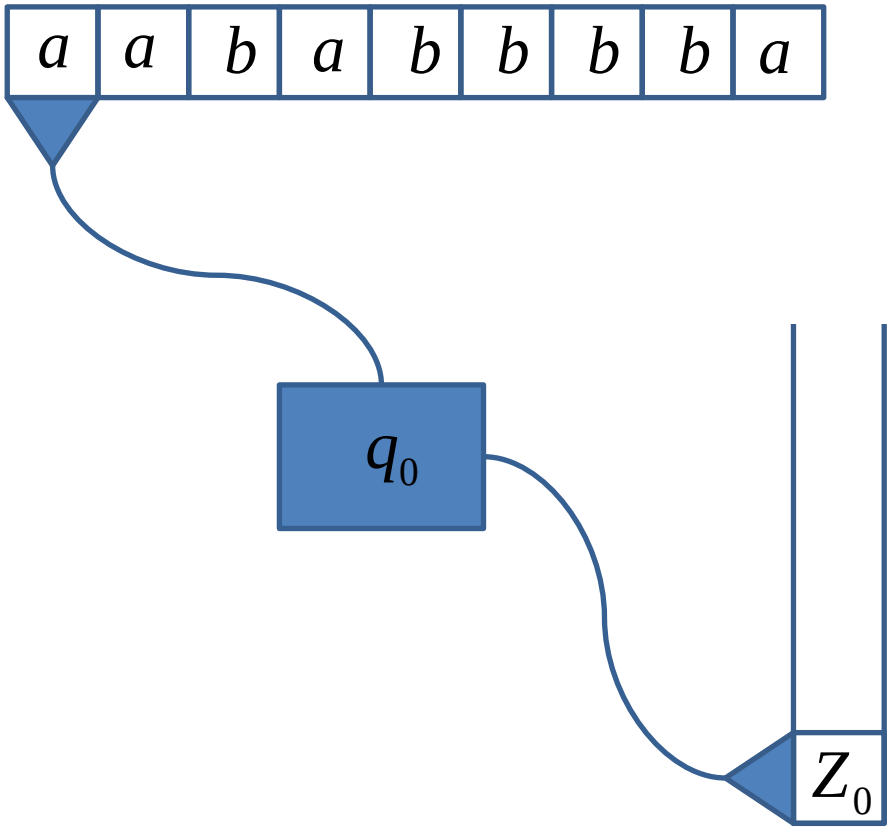
Zásobníkový automat je šestice $A = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$, kde

- Q je konečná množina stavů
- Σ je vstupní abeceda
- Γ je zásobníková abeceda
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to P(Q \times \Gamma^*)$ je přechodová relace
- $q_0 \in Q$ je počáteční (iniciální) stav
- $Z_0 \in \Gamma$ je počáteční zásobníkový symbol

Jedná se o nedeterministický model.

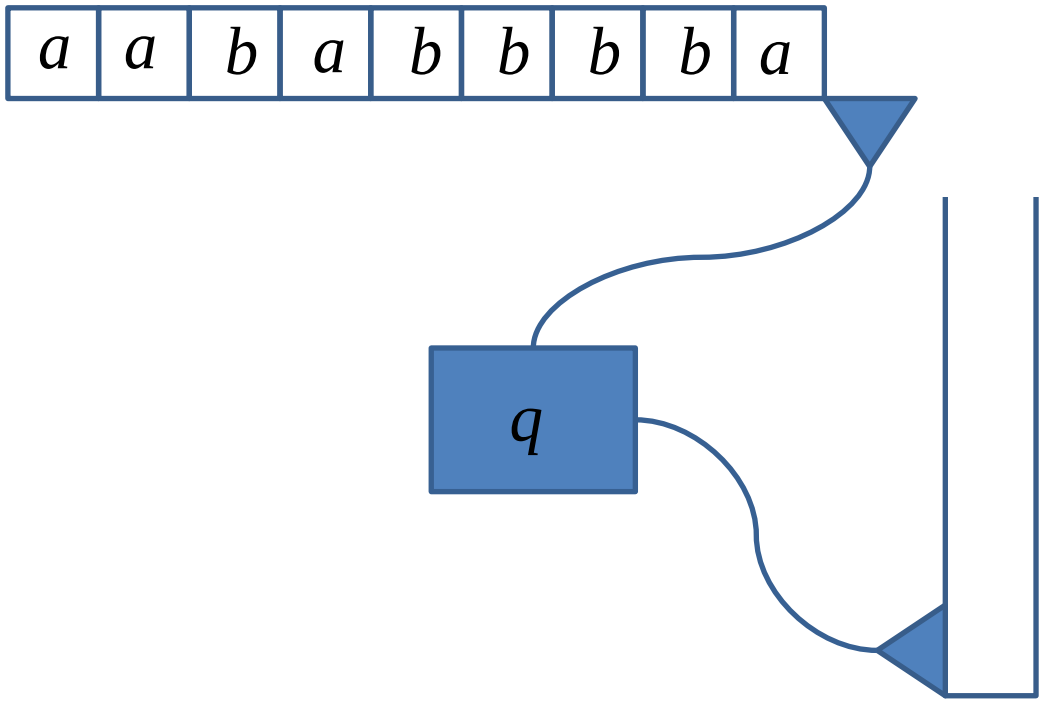
Zásobníkový automat – počáteční stav

- řídící jednotka v počátečním stavu q_0
- zásobník obsahuje pouze počáteční symbol Z_0



Přijmutí vstupu

Přijímání prázdným zásobníkem – po přečtení celého vstupu je zásobník prázdný.

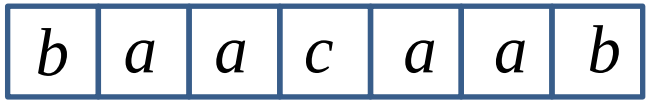


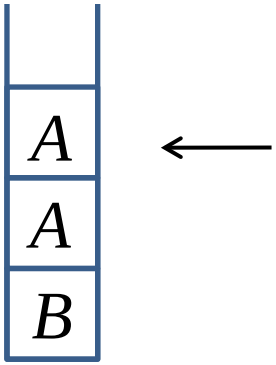
Příklad zásobníkového automatu I

Sestrojíme zásobníkový automat přijímající jazyk

$$L_1 = \{wcw^R \mid w \in \{a,b\}^*\}$$

vstup:



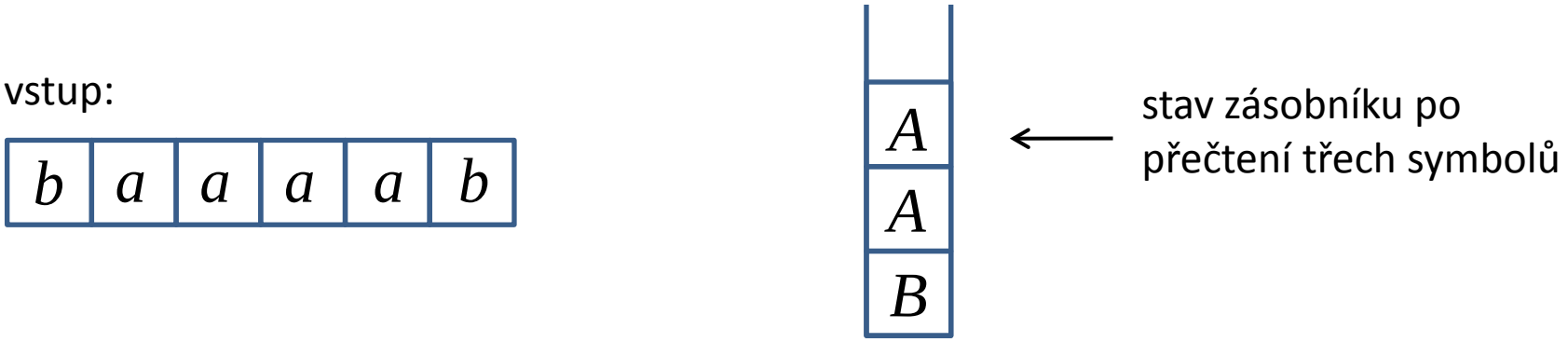


stav zásobníku před přečtením symbolu *c*

- Levou část postupně reprezentujeme v zásobníku.
- Po přečtení symbolu *c* kontrolujeme, zda obsah zásobníku se shoduje s pravou částí. Zkontrolované symboly ze zásobníku mažeme.

Příklad zásobníkového automatu II

Modifikace předchozího jazyka: $L_2 = \{ww^R \mid w \in \{a,b\}^*\}$



- Sestrojíme (nedeterministický) zásobníkový automat: pro každý vstupní znak nedeterministicky testuje, zda se jedná o konec první poloviny vstupu.

Ekvivalence automatů a gramatik

Věta: Ke každému zásobníkovému automatu lze sestrojit bezkontextovou gramatiku, která generuje stejný jazyk.

Zás. automat můžeme převést na ekvivalentní s jediným stavem (zapisujeme původní stavy na zásobník)

$A = (\{q_0\}, \Sigma, \Gamma, \delta, q_0, Z_0)$ Σ, Γ disjunktní

$G = (\Gamma, \Sigma, Z_0, P)$

$(N \to a\alpha) \in P \Leftrightarrow (q_0, \alpha) \in \delta(q_0, a, N)$ $a \in (\Sigma \cup \{\varepsilon\})$

Ekvivalence automatů a gramatik

Věta: Ke každé bezkontextové gramatice lze sestrojit zásobníkový automat, který rozpoznává stejný jazyk.

$$G = (\Pi, \Sigma, S, P)$$

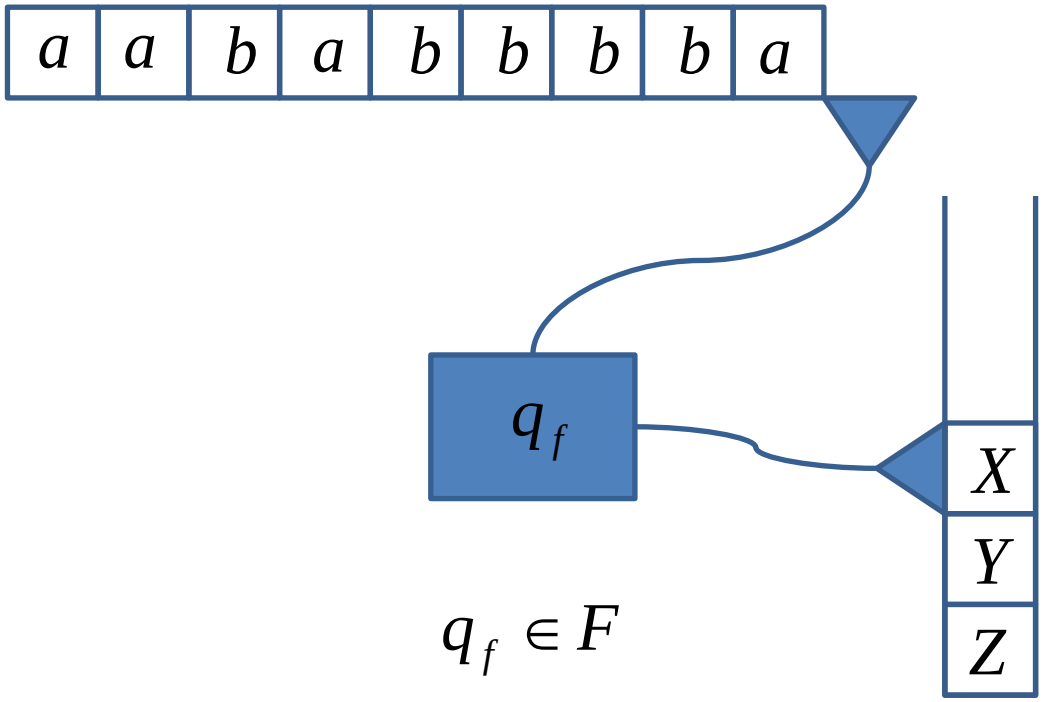
$$A = (\{q_0\}, \Sigma, \Pi \cup \Sigma, \delta, q_0, S)$$

$$\forall N \in \Pi : \delta(q_0, \varepsilon, N) = \{(q_0, \alpha) \mid (N \to \alpha) \in P\}$$

$$\forall a \in \Sigma : \delta(q_0, a, a) = \{(q_0, \varepsilon)\}$$

Přijímání koncovým stavem

Přijímání koncovým stavem – rozšíříme definici o množinu přijímacích stavů $F \subseteq Q$



- je ekvivalentní s přijímáním prázdným zásobníkem

Deterministické zásobníkové automaty

$A = (Q, \Sigma, \Gamma, \delta, q_0, F, Z_0)$

$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to P(Q \times \Gamma^*)$

1. $|\delta(q, a, X)| \leq 1$ pro všechna $a \in \Sigma \cup \{\varepsilon\}$
2. je-li $|\delta(q, \varepsilon, X)| = 1$, pak $|\delta(q, a, X)| = 0$ pro $\forall a \in \Sigma$

Přijímání definováno koncovým stavem.

Deterministické zásobníkové automaty nerozpoznají všechny bezkontextové jazyky, odlišujeme deterministické bezkontextové jazyky.

$L_2 = \{ww^R \mid w \in \{a, b\}^*\}$ .. není deterministický bezkontextový

LL(k) a LR(k) syntaktické analyzátory

- automaty pro parsování deterministických bezkontextovým jazykům, odvozené ze zásobníkových automatů

LL(k) ← velikost výhledu
↑ ↖ leftmost derivation
left to right – směr pohybu hlavy

LR(k)
↖ rightmost derivation

LL(k) jsou slabší než LR(k)

LR(1) .. parsují všechny det. bezkontextové jazyky
LR(0) .. parsují všechny det. bezkontextové jazyky, pokud je vstup ohraničen zprava speciálním znakem

GNU bison, Yacc – generátory analyzátorů

Konstrukce LL(1) analyzátoru

1. Pro každé pravidlo $A \to w$ definujeme množiny *First(A)* a *First(w)* obsahující terminály, kterými mohou začínat slova vygenerovaná z *A*, resp. *w*.

2. Pro každý neterminál A definujeme množinu *Follow(A)* obsahující terminály, které mohou následovat bezprostředně za *A* v nějakém odvození (např. *a* následuje za *A* v řetězci *uAav*).

Pozn.: *u,v,w* označují řetězce z terminálů a neterminálů.

Příklad gramatiky

$$S \rightarrow F$$

$$S \rightarrow (S + F)$$

$$F \rightarrow a$$

$$First(S) = \{(, a\} \qquad First((S + F)) = \{(\}$$

$$First(F) = \{a\} \qquad First(a) = \{a\}$$

$$Follow(S) = \{+,)\}$$

$$Follow(F) = \{)\}$$

Sestavení parsovací tabulky

T[A,a] obsahuje pravidlo $A \to w$ právě tehdy když

- *a* je ve *First(w)* nebo
- ε je ve *First(w)* a *a* je ve *Follow(A)*

	(	)	a	+	$
S	$S \to (S+F)$		$S \to F$		
F			$F \to a$		

Vstup je ukončen znakem *$*.

Výpočet množiny First

Následující kroky provádíme pro všechna pravidla gramatiky, $A \to w$ označuje libovolné pravidlo.

1. Inicializuj každé *First(A)* a *First(w)* prázdnou množinou.
2. Spočítej *First(w)* následovně:
   - *First(av)* = *{a}* pro každý terminál *a*
   - *First(Nv)* = *First(N)* pro každý neterminál *N* kde ε není ve *First(N)*
   - *First(Nv)* = *First(N)* – *{*ε*}* $\cup$ *First(v)* pro každý neterminál N jehož *First(N)* obsahuje ε
   - *First(*ε*)* = *{*ε*}*
3. Vlož prvky z *First(w)* do *First(A)*.
4. Opakuj kroky 2 a 3. Skonči v případě, že se žádná množina již nezměnila.

Výpočet množiny Follow

1. Inicializuj každé *Follow(A)* prázdnou množinou.
2. Pro každé pravidlo tvaru $A \to uBv$
   - každý terminál z *First(v)* přidej do *Follow(B)*
   - když *First(v)* obsahuje ε, vlož vše z *Follow(A)* do *Follow(B)*
3. Opakuj krok 2 dokud se některé množiny *Follow* mění.

Obecný parsovací algoritmus

Cocke-Younger-Kasami (CYK) algoritmus

- využívá techniku dynamického programování

$G = (\Pi, \Sigma, S, P)$ - bezkontextová gramatika v Chomského normální formě

Vstup: $u = a_1 a_2 \ldots a_n$ (slovo nad Σ)

Otázka: Generuje gramatika *G* slovo *u* ?

$\Pi = \{N_1, \ldots, N_m\}$ a $S = N_1$ (očíslujeme neterminály)

Počítáme boolovské pole $g[1..n, 1..n, 1..m]$, kde hodnotu $g[i, j, k]$ nastavíme na *true*, pokud podslovo slova *u* začínající indexem *i* a mající délku *j* lze vygenerovat z neterminálu N_k.

CYK algoritmus

```
boolean CYK(G,u)
  for i:=1 to n do
    for each N_j → a_i do
      g[i,1,j]:=true;
  for i:=2 to n do // délka podslova
    for j:=1 to n-i+1 do // index podslova
      for k:=1 to i-1 do  // pozice rozdělení podslova
        for each N_p → N_q N_r do
          if g[j,k,q] && g[j+k,i-k,r] then
            g[j,i,p]:=true;
  return g[1,n,1];
end
```

Příklad

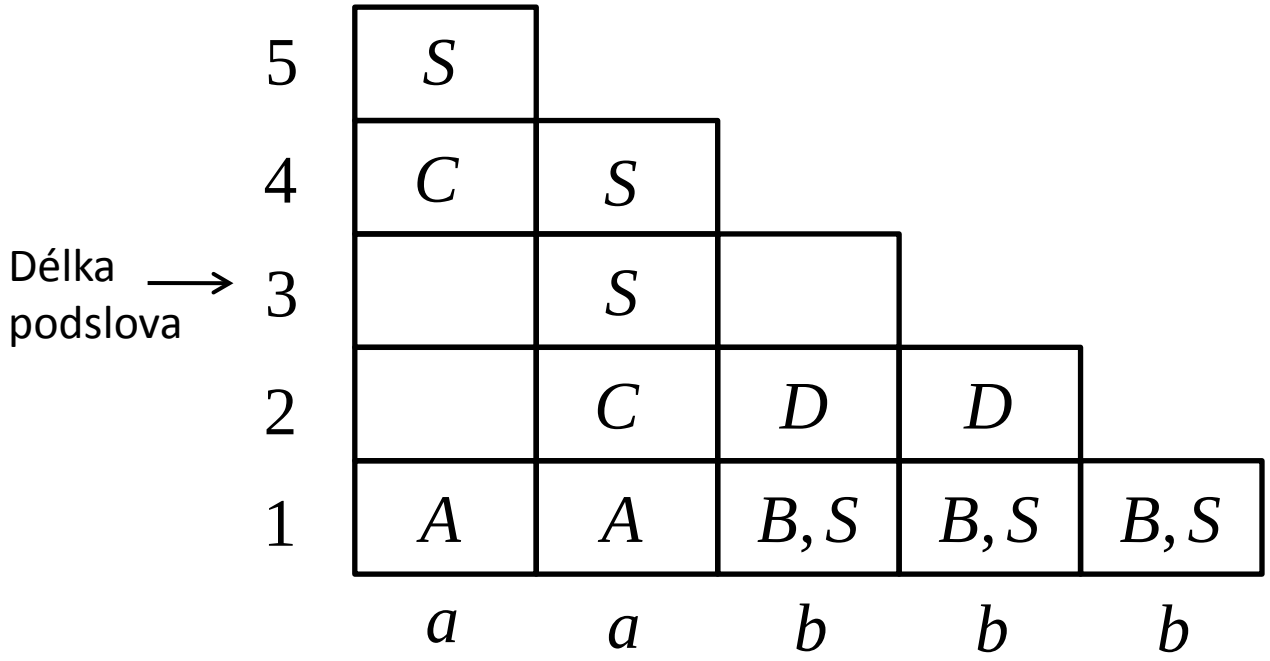
$G = (\Pi, \Sigma, S, P)$ $\quad \Sigma = \{a, b\}$ $\quad \Pi = \{S, A, B, C, D\}$

$S \to CB$ $\quad D \to BB$ $\quad A \to a$

$S \to CD$ $\quad C \to AS$ $\quad B \to b$

$S \to b$

$u = aabbb$



Získání derivačního stromu

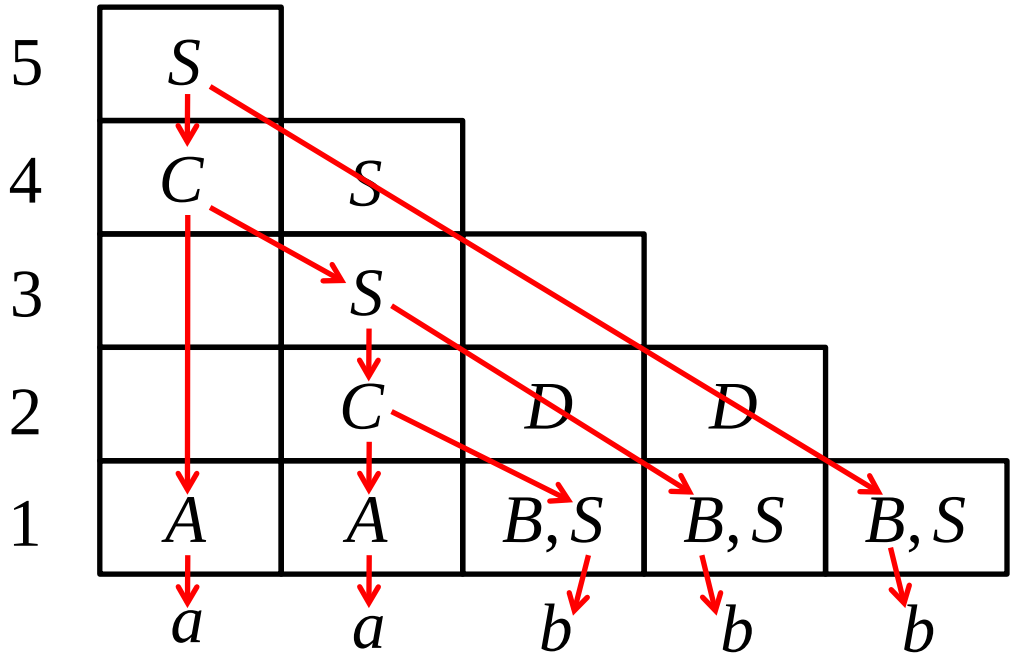
$G = (\Pi, \Sigma, S, P)$ $\quad \Sigma = \{a, b\}$ $\quad \Pi = \{S, A, B, C, D\}$

$S \to CB$ $\quad D \to BB$ $\quad A \to a$

$S \to CD$ $\quad C \to AS$ $\quad B \to b$

$S \to b$

$u = aabbb$



Časová složitost CYK algoritmu

$G = (\Pi, \Sigma, S, P)$

$u = a_1 a_2 \ldots a_n$

V algoritmu jsou 4 vnořené cykly

=> časová složitost je $O(|P| \cdot n^3)$