Syntaktická analýza, Top-down prístup, LL(1)

Ing. Viliam Hromada, PhD.

C-510
Ústav informatiky a matematiky
FEI STU

viliam.hromada@stuba.sk

Syntaktická analýza

Cieľom syntaktickej analýzy je:

- Zistiť, či dané slovo (pri kompilátoroch program) patrí do jazyka generovaného danou bezkontextovou gramatikou (pri kompilátoroch či je program syntakticky správny).
- V prípade, že slovo patrí do jazyka je ďalšou úlohou nájsť jeho strom odvodenia - pri kompilátoroch sa podľa neho ďalej vykonáva sémantická analýza - interpretácia, *čo má program robiť*.

- Syntaktickú analýzu, teda proces zistenia, či má daná postupnosť terminálnych symbolov v gramatike deriváciu - a ak áno, ako derivácia vyzerá - vykonávajú tzv. **syntaktické analyzátory**.
- Prakticky použiteľné syntaktické analyzátory majú lineárnu zložitosť spracovania vstupu vzhľadom na jeho dĺžku, $O(n)$, kde n je dĺžka reťazca, ktorého deriváciu hľadajú.
- Tento proces sa nazýva aj *parsing* (parsovanie).
- Rozhodovací problém, pri ktorom je daná gramatika $G = (N, T, P, S)$, reťazec $w \in T^*$ a chceme zistiť, či $w \in L(G)$ sa nazýva aj tzv. *parsing problem*, t.j. parsovací problém.

Syntaktický analyzátor

- **Vstupom** syntaktického analyzátora je **postupnosť lexém** rozpoznaná lexikálnym analyzátorom - tie predstavujú terminálne symboly gramatiky popisujúcej počitačový jazyk.
- **Výstupom** syntaktického analyzátora je **postupnosť pravidiel**, aplikáciou ktorých dostaneme strom odvodenia programu (vstupného slova), ktorý predstavuje **syntaktickú reprezentáciu** programu.
- Z praktických dôvodov je dobré, aby bol strom odvodenia jednoznačný; navyše musí byť v postupnosti pravidiel jednoznačne určené, na ktorý neterminál sa pravidlo aplikuje (často jednoduché, ak sa povie, že sa jedná o ľavé/pravé odvodenia)

Syntaktická analýza zhora-nadol (top-down)

- Strom odvodenia sa konštruuje od **koreňa k listom**, podľa terminálov príslušného slova čítaných zľava-doprava.
- Odvodenie slova v danej gramatike sa v tomto prípade deje od počiatočného neterminálu k terminálom **ľavou deriváciou**.
- Postupnosť pravidiel generovaná syntaktickým analyzátorom sa teda bude vzťahovať na **ľavé odvodenie vstupného slova**.

Príklad

Gramatika $G = (\{$ <program>, <príkaz>, <príkazy>$\}$, $\{$**begin, end, p, ;** $\}, P,$<program>), pravidlá P:

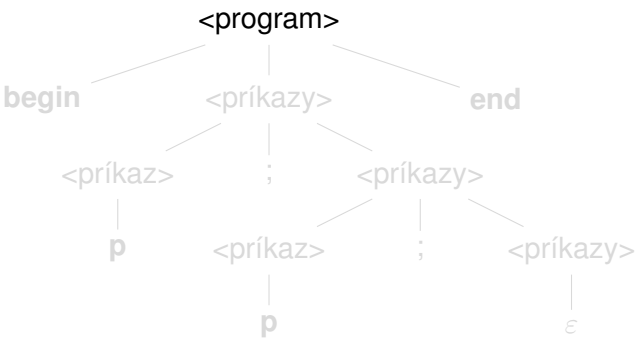
<program> $\rightarrow$ **begin**<príkazy>**end**

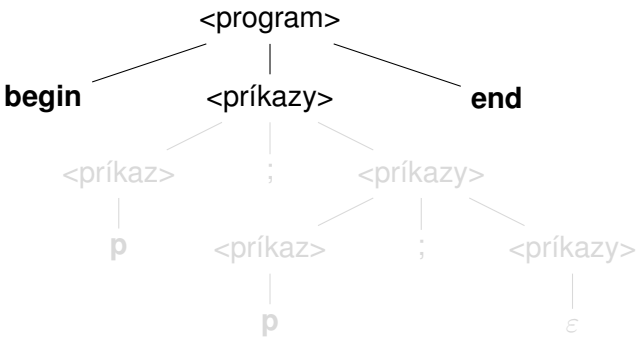
<príkazy> $\rightarrow$ <príkaz> ; <príkazy>

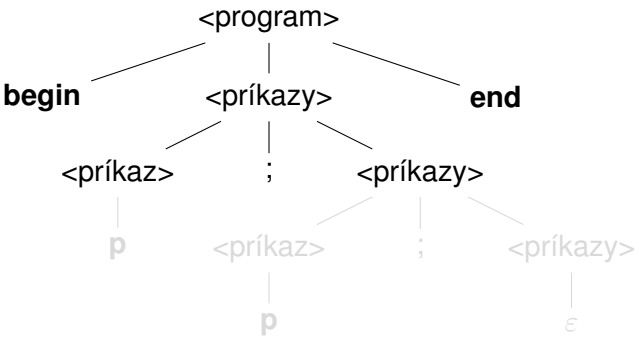
<príkazy> $\rightarrow \varepsilon$

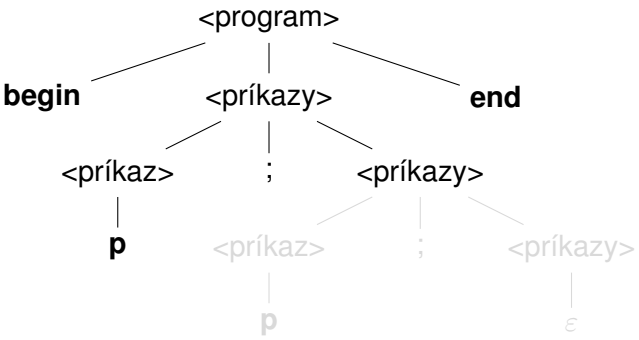
<príkaz> $\rightarrow$ **p**

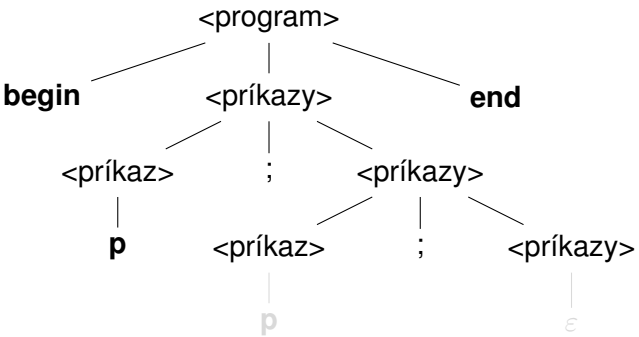
Uvažujme odvodenie vety **begin p; p; end** od koreňa (počiatočného neterminálu) k listom (terminálom) tak, že vždy nahradíme prvý neterminál zľava (ľavá derivácia):

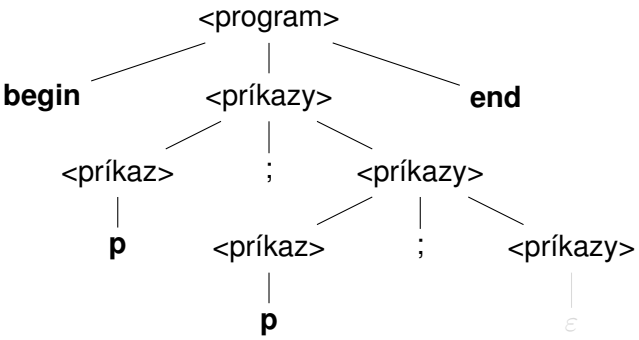

<program>
begin
<príkazy>
end
<príkaz>
;
<príkazy>
p
<príkaz>
;
<príkazy>
p
ε

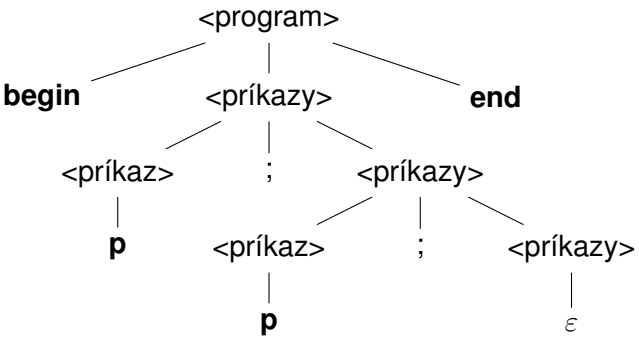

<program>
begin
<príkazy>
end
<príkaz>
;
<príkazy>
p
<príkaz>
;
<príkazy>
p
ε


<program>
begin
<príkazy>
end
<príkaz>
;
<príkazy>
p
<príkaz>
;
<príkazy>
p
ε


<program>
begin
<príkazy>
end
<príkaz>
;
<príkazy>
p
<príkaz>
;
<príkazy>
p
ε


<program>
begin
<príkazy>
end
<príkaz>
;
<príkazy>
p
<príkaz>
;
<príkazy>
p
ε


<program>
begin
<príkazy>
end
<príkaz>
;
<príkazy>
p
<príkaz>
;
<príkazy>
p
ε


<program>
begin
<príkazy>
end
<príkaz>
;
<príkazy>
p
<príkaz>
;
<príkazy>
p
ε

Príklad (pokr.)

Tomu zodpovedá nasledovné odvodenie:

<program> $\Rightarrow_l$ **begin**<príkazy>**end** $\Rightarrow_l$
$\Rightarrow_l$ **begin** <príkaz> ; <príkazy> **end** $\Rightarrow_l$
$\Rightarrow_l$ **begin p** ; <príkazy> **end** $\Rightarrow_l$
$\Rightarrow_l$ **begin p** ; <príkaz> ; <príkazy> **end** $\Rightarrow_l$
$\Rightarrow_l$ **begin p ; p ;** <príkazy> **end** $\Rightarrow_l$
$\Rightarrow_l$ **begin p ; p ; end**

Top-down analýza

- Uvedený postup hľadania derivačného stromu od koreňa k listom, pričom sa vlastne vytvára ľavá derivácia vstupného slova, sa dá modelovať pomocou nedeterministického zásobníkového automatu.

Teoretický model top-down analyzátora

Top-down syntaktický analyzátor pre bezkontextovú gramatiku $G = (N, T, P, S)$ bude (nedeterministický) zásobníkový automat, ktorý pracuje na základe nasledovných princípov:

- Automat v zásobníku simuluje ľavé odvodenie slova w, t.j. zásobníkové symboly sú terminály a neterminály gramatiky G.
- Počiatočný zásobníkový symbol bude začiatočný neterminál S gramatiky G.

Teoretický model top-down analyzátora (pokr.)

- Kým sa automat nezasekne opakuje nasledovné činnosti:
  1. **Expanzia** - ak sa na vrchu zásobníka nachádza neterminál $A \in N$, algoritmus nedeterministicky vyberie jedno z $A-$pravidiel $A \to \alpha$ a neterminál A nahradí pravou stranou α.
  2. **Porovnanie** - ak sa na vrchu zásobníka nachádza terminál $a \in T$, porovná sa so symbolom na vstupe t; ak sa zhodujú, automat odstráni symbol a z vrchu zásobníka a prejde na ďalší vstupný symbol; ak sa nezhodujú, automat sa zasekne, slovo neakceptuje (=syntaktická chyba).
- Automat akceptuje vstupné slovo (=syntakticky správne), ak spracuje celý vstup a skončí s prázdnym zásobníkom.
- Ak automat vždy pošle na výstup pravidlo, podľa ktorého vykonal expanziu, získa sa postupnosť pravidiel, ktorá dáva ľavú deriváciu slova w v gramatike G.

Syntaktická analýza top-down I

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$, reťazec w
Výstup: Zistenie, či $w \in L(G)$ a postupnosť pravidiel ľavého odvodenia w

```
 1: vlož do zásobníka S
 2: pokiaľ zásobník nie je prázdny rob
 3:    ak na vrchu zásobníka je A ∈ N potom
 4:       odstráň z vrchu zásobníka A
 5:       vlož do zásobníka pravú stranu α niektorého z pravidiel A → α
 6:       pošli na výstup pravidlo A → α
 7:    koniec ak
 8:    ak na vrchu zásobníka je a ∈ T potom
 9:       ak a je totožné s aktuálnym vstupným symbolom t potom
10:          odstráň a zo zásobníka
11:          prejdi na ďalší vstupný symbol t
12:       inak
13:          neakceptuj vstupné slovo a skonči
```

Syntaktická analýza top-down II

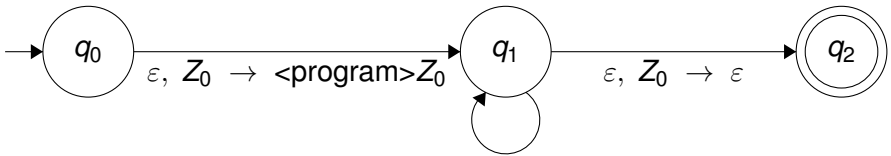
14: **koniec ak**
15: **koniec ak**
16: **koniec pokiaľ**
17: **ak** bolo spracované celé vstupné slovo **potom**
18: akceptuj vstupné slovo
19: **inak**
20: neakceptuj vstupné slovo
21: **koniec ak**

Príklad

Uvažujme gramatiku zo slajdu č. 6 a vetu **begin p ; p ; end**.

r.	Zvyšok vstupu	Zásobník ☐	Akcia
1	**begin p; p; end**	<program>	E1
2	**begin p; p; end**	**begin**<príkazy>**end**	P
3	**p; p; end**	<príkazy>**end**	E2
4	**p; p; end**	<príkaz>;<príkazy>**end**	E4
5	**p; p; end**	**p;**<príkazy>**end**	P
6	**; p; end**	;<príkazy>**end**	P
7	**p; end**	<príkazy>**end**	E2
8	**p; end**	<príkaz>;<príkazy>**end**	E4
9	**p; end**	**p**; <príkazy>**end**	P
10	**; end**	; <príkazy>**end**	P
11	**end**	<príkazy>**end**	E3
12	**end**	**end**	P
13	ε	ε	A

Ekvivalentný ZA



ε, <program> $\rightarrow$ **begin**<prikazy>**end**

ε, <prikazy> $\rightarrow$ <prikaz>**;**<prikazy>

ε, <prikazy> $\rightarrow \varepsilon$

ε, <prikaz> $\rightarrow$ **p**

begin, **begin** $\rightarrow \varepsilon$

end, **end** $\rightarrow \varepsilon$

p, **p** $\rightarrow \varepsilon$

;, **;** $\rightarrow \varepsilon$

Top-down analýza $\{w \in \{a, b\}^* \mid \sharp_a(w) = \sharp_b(w)\}$

Gramatika pre uvedený jazyk:

1. $S \to SaSbS$
2. $S \to SbSaS$
3. $S \to \varepsilon$

Príklad - výpočet *abaabb*

Uvažujme gramatiku zo slajdu č. 16 a reťazec *abaabb*. Potom akceptujúci výpočet ZA bude (zápis vo forme tabuľky):

r.	Zvyšok vstupu	Zásobník ⊐	Akcia
1	*abaabb*	*S*	E1
2	*abaabb*	*SaSbS*	E3
3	*abaabb*	*aSbS*	P
4	*baabb*	*SbS*	E3
5	*baabb*	*bS*	P
6	*aabb*	*S*	E1
7	*aabb*	*SaSbS*	E3
8	*aabb*	*aSbS*	P
9	*abb*	*SbS*	E1
10	*abb*	*SaSbSbS*	E3
11	*abb*	*aSbSbS*	P
12	*bb*	*SbSbS*	E3

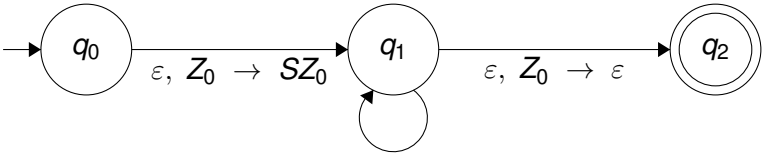
Príklad - výpočet *abaabb*

r.	Zvyšok vstupu	Zásobník ⊐	Akcia
13	*bb*	*bSbS*	P
14	*b*	*SbS*	E3
15	*b*	*bS*	P
16	ε	*S*	E3
17	ε	ε	A

Príklad - výpočet *abb*

r.	Zvyšok vstupu	Zásobník ☐	Akcia
1	*abb*	*S*	E1
2	*abb*	*SaSbS*	E3
3	*abb*	*aSbS*	P
4	*bb*	*SbS*	E3
5	*bb*	*bS*	P
6	*b*	*S*	E2
7	*b*	*SbSaS*	E3
8	*b*	*bSaS*	P
9	ε	*SaS*	E3
10	ε	*aS*	ERROR

Pre *abb* by aj všetky iné možné výpočty skončili s chybou - slovo *abb* teda nepatrí do $L(G)$.



$$\varepsilon,\ S \rightarrow SaSbS$$
$$\varepsilon,\ S \rightarrow SbSaS$$
$$\varepsilon,\ S \rightarrow \varepsilon$$
$$a,\ a \rightarrow \varepsilon$$
$$b,\ b \rightarrow \varepsilon$$

Vlastnosti teoretického modelu

- Predstavený model je použiteľný s ľubovoľnou bezkontextovou gramatikou. Dokáže nájsť derivačný strom pre slová generované aj nejednoznačnými, aj nedeterministickými gramatikami.
- **Nedeterminizmus** teoretického modelu je však prekážkou pri jeho praktickej použiteľnosti, pretože pri hľadaní derivácie / spracovaní slova je potrebné sledovať všetky možné vetvy výpočtu.
  - V príklade na slajde 14 je nedeterminizmus na riadkoch 3,7,11, kde sa vyberalo z 2 pravidiel na expanziu neterminálu <príkazy>.
- Ako teda vytvoriť prakticky použiteľný (t.j. deterministický) top-down parser?

Syntaktický analyzátor jazyka LL(1)

Aby bol syntaktický analyzátor použiteľný v praxi, je potrebné, aby vedel **deterministicky** spracovať vstupné reťazce s **lineárnou** časovou zložitosťou.

T.j. musíme odstrániť nedeterminizmus z teoretického modelu.

Daný model analyzátora obsahuje jediný nedeterminizmus - keď je potrebné expandovať neterminál A na vrchu zásobníka a príslušná gramatika obsahuje viacero pravidiel s neterminálom A na ľavej strane.

V praxi sa nedeterminizmus odstráni tak, že automat si príslušné pravidlo **vyberie** na základe **aktuálneho vstupného symbolu** t.

Takýto analyzátor má názov LL(1) = left-left-1. T.j. vstupný reťazec sa spracúva **zľava**, vytvára sa **ľavé** odvodenie a na rozhodovanie sa používa 1 vstupný symbol.

Syntaktický analyzátor jazyka LL(1)

Definícia

Nech $G = (N, T, P, S)$ je bezkontextová gramatika. ***Rozkladová tabuľka syntaktického analyzátora LL(1)*** *$RT[A, t]$ je zobrazenie $N \times T_\varepsilon \to P$ popisujúce, ktoré pravidlo sa použije na expanziu neterminálu A nachádzajúceho sa na vrchu zásobníka, ak aktuálny vstupný symbol je t (ak $t = \varepsilon$, potom bol prečítaný celý vstupný reťazec).*

Rozkladová tabuľka je zvyčajne neúplné zobrazenie - ak neexistuje v tabuľke záznam pre aktuálny neterminál A a symbol t, znamená to **syntaktickú chybu**.

Syntaktický analyzátor jazyka LL(1) I

Vstup: Bezkontextová gramatika $G = (N, T, P, S)$, reťazec w, rozkladová tabuľka syntaktického analyzátora LL(1) RT

Výstup: Zistenie, či $w \in L(G)$ a postupnosť pravidiel ľavého odvodenia w

```
1:  vlož do zásobníka S
2:  t ← prvá lexéma zo vstupu
3:  pokiaľ zásobník nie je prázdny rob
4:      X ← symbol z vrchu zásobníka a zároveň ho odstráň zo zásobníka
5:      ak X ∈ N a zároveň RT[X, t] = X → α potom
6:          vlož do zásobníka α aby ľavý krajný symbol bol na vrchu
7:          pošli na výstup pravidlo X → α
8:      inak
9:          ak X ∈ T a zároveň X = t potom
10:             t ← ďalšia lexéma zo vstupu
11:         inak
12:             spracuj syntaktickú chybu
```

Syntaktický analyzátor jazyka LL(1) II

13: **koniec ak**
14: **koniec ak**
15: **koniec pokiaľ**
16: akceptuj vstupné slovo

Ako vytvoriť rozkladovú tabuľku - idea

- Ak je na vrchu zásobníka neterminál A a na vstupe terminál t, pýtame sa, za akých podmienok použiť pravidlo $A \to \alpha$, aby sa automat nezasekol.
- Automat sa zasekne v momente, keď sa vstupný terminál nezhoduje s terminálom na vrchu zásobníka.
- Preto pri výbere pravidla do RT postupujeme tak, aby sme mali zabezpečené, že po expanzii neterminálu A zvoleným pravidlom $A \to \alpha$ bude vstupný symbol t rovný prvému terminálu, ktorý sa môže zjaviť na vrchu zásobníka.
- Ak expandujeme podľa pravidla $A \to \alpha$, tak na vrchu zásobníka sa môže zjaviť len terminál patriaci do množiny:
  - $FIRST(\alpha) - \{\varepsilon\}$
  - ak $\varepsilon \in FIRST(\alpha)$ tak $FOLLOW(A)$.

Ako vytvoriť rozkladovú tabuľku - množiny *PREDICT*

Definícia

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

$$PREDICT(A \to \alpha) =$$
$$= \begin{cases} FIRST(\alpha), & \text{ak } \varepsilon \notin FIRST(\alpha) \\ (FIRST(\alpha) - \{\varepsilon\}) \cup FOLLOW(A), & \text{ak } \varepsilon \in FIRST(\alpha) \end{cases}$$

T.j. množina *PREDICT* signalizuje, ktoré pravidlo použiť na expanziu *A*. Samozrejme, ak sa máme **jednoznačne** rozhodnúť, musí platiť: Ak $A \to \alpha_i$ a $A \to \alpha_j$ sú 2 rôzne *A*-pravidlá, potom:

$$PREDICT(A \to \alpha_i) \cap PREDICT(A \to \alpha_j) = \emptyset$$

Definícia

Redukovaná gramatika $G = (N, T, P, S)$ v ktorej všetky dvojice pravidiel majú prázdny prienik množín PREDICT, sa nazýva ***LL(1) gramatika****.*

Ak je teda gramatika typu $LL(1)$, potom RT zostrojíme nasledovne:

$$\text{Ak } t \in PREDICT(A \to \alpha), \text{ potom } RT[A, t] \text{ obsahuje pravidlo } A \to \alpha.$$

Ak je gramatika typu $LL(1)$, každá položka rozkladovej tabuľky obsahuje maximálne jedno pravidlo a rozhodovanie o použití pravidla pri expanzii je jednoznačné.

Príklad zostrojenia RT

Zostrojte RT pre gramatiku:

1. <program> → **begin**<príkazy>**end**
2. <príkazy> → <príkaz> ; <príkazy>
3. <príkazy> → ε
4. <príkaz> → **p**

Najprv zostrojíme množiny *FIRST* a *FOLLOW*:

FIRST	<program>	<príkazy>	<príkaz>	**begin**	**end**	;	**p**
Celkom	**begin**	**p**, ε	**p**	**begin**	**end**	;	**p**

FOLLOW	<program>	<príkazy>	<príkaz>
Celkom:	ε	**end**	;

Príklad zostrojenia RT (pokr.)

Výsledné množiny *PREDICT* pre pravidlá:

- *PREDICT*(1.) = *FIRST*(**begin**<príkazy>**end**) = {**begin**}.
- *PREDICT*(2.) = *FIRST*(<príkaz>;<príkazy>) = {**p**}
- *PREDICT*(3.) = *FOLLOW*(<príkazy>) = {**end**}
- *PREDICT*(4.) = *FIRST*(**p**) = {**p**}.

T.j. rozkladová tabuľka:

RT	**begin**	**end**	;	**p**	ε
<program>	1.				
<príkazy>		3.		2.	
<príkaz>				4.	

Keďže každá položka obsahuje max. 1 pravidlo, gramatika je naozaj LL(1) gramatika.

Príklad zostrojenia RT

Zostrojte RT pre gramatiku:

1. $S \rightarrow SaSbS$
2. $S \rightarrow SbSaS$
3. $S \rightarrow \varepsilon$

Najprv zostrojíme množiny *FIRST* a *FOLLOW*:

FIRST	S	a	b
Celkom	$\{\varepsilon, a, b\}$	$\{a\}$	$\{b\}$

FOLLOW	S
Celkom:	$\{\varepsilon, a, b\}$

Príklad zostrojenia RT (pokr.)

Výsledné množiny *PREDICT* pre pravidlá:

- $PREDICT(1.) = FIRST(SaSbS) = \{a, b\}$.
- $PREDICT(2.) = FIRST(SbSaS) = \{a, b\}$
- $PREDICT(3.) = (FIRST(\varepsilon) \setminus \{\varepsilon\}) \cup FOLLOW(S) =$
  $= FOLLOW(S) = \{a, b, \varepsilon\}$

T.j. rozkladová tabuľka:

RT	a	b	ε
S	1., 2., 3.	1., 2., 3.	3.

Keďže existujú položky s viac ako 1 pravidlom, gramatika **nie je** LL(1) gramatika.

Poznámky

- LL(1) analyzátory/jazyky sa dajú rozšíriť na rozhodovanie o expanzii na základe k vstupných symbolov, $k \geq 1$. Vtedy hovoríme o $LL(k)$ gramatikách/analyzátoroch.
- Problémom je, že pre niektoré bezkontextové gramatiky sa nedá zostrojiť ani $LL(1)$ analyzátor, ani $LL(k)$ analyzátor.
- V niektorých prípadoch je riešením pokúsiť sa nájsť ekvivalentnú gramatiku pre daný jazyk, ktorá $LL(1)$ je.

Príklad

Gramatika, ktorá generuje jazyk $\{w \in \{a, b\}^* \mid \sharp_a(w) = \sharp_b(w)\}$:

1. $S \to SaSbS$
2. $S \to SbSaS$
3. $S \to \varepsilon$

nie je $LL(1)$

Ten istý jazyk, t.j. $\{w \in \{a,b\}^* \mid \sharp_a(w) = \sharp_b(w)\}$, sa však predsa len dá generovať $LL(1)$ gramatikou, konkrétne:

1. $S \to aBS \mid bAS$
2. $A \to a \mid bAA$
3. $B \to b \mid aBB$

Problém if-then-else pri syntaktickej analýze jazykov LL

Tento problém sa vyskytuje u jazykov, kde neexistuje explicitný ukončovací symbol podmieneného príkazu (napr. C, Java). Typická gramatika popisujúca štruktúru podmienených príkazov môže byť:

1. <príkaz s ;> → <príkaz>;
2. <príkaz> → `if`<podmienka>`then`<príkaz><else časť>
3. <príkaz> → `iný`
4. <else časť> → `else`<príkaz>
5. <else časť> → ε
6. <podmienka> → `p`.

Rozkladová tabuľka danej gramatiky

Daná gramatika pre if-then-else nie je *LL*(1) gramatika, pretože:

RT	`if`	`then`	`else`	`iný`	`;`	`p`
<príkaz s ;>	1.			1.		
<príkaz>	2.			3.		
<else časť>			4. 5.		5.	
<podmienka>						6.

Keďže väčšinou sa `else` viaže s najbližším príkazom `if`, tak sa dá pravidlu č. 4 prednosť pred pravidlom číslo 5, ak nastane situácia, že na vrchu zásobníka je symbol <else časť> a na vstupe terminál `else`.

Použitá literatúra

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

Dedera, Ľ: *Počítačové jazyky a ich spracovanie*.

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