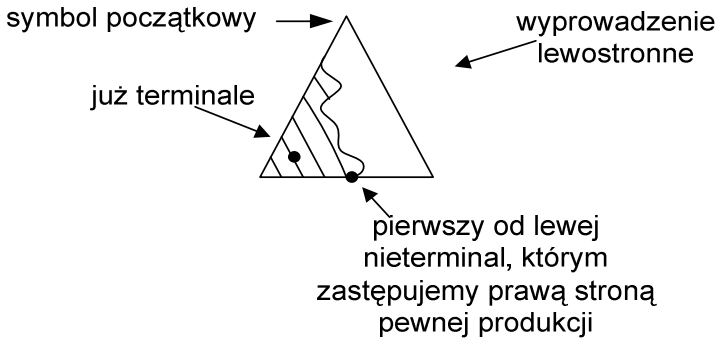
AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Parsery LL(1)

Teoria kompilacji

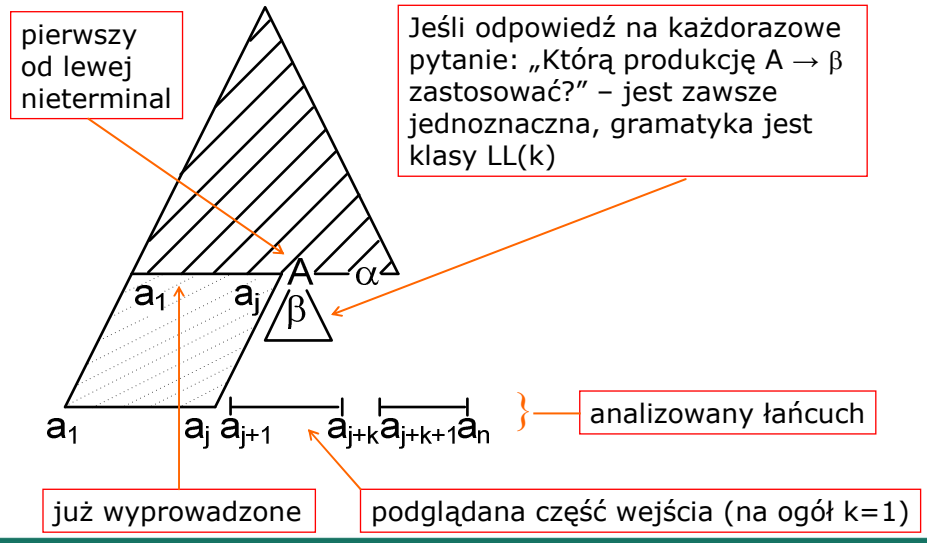
Dr inż. Janusz Majewski
Katedra Informatyki

Zadanie analizy generacyjnej (zstępującej, top-down)

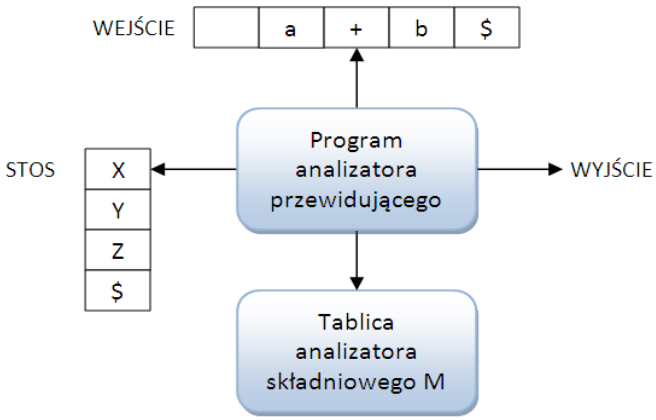


Odtworzenie wywodu lewostronnego metodą top-down

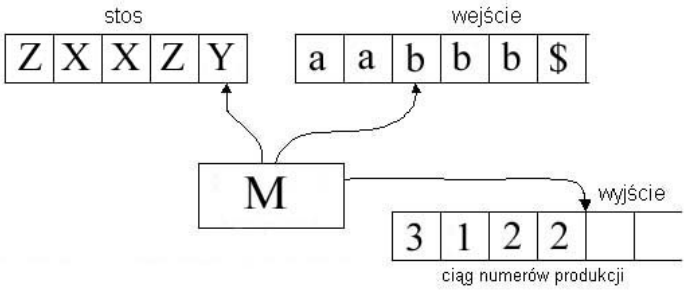
Istota gramatyki i parsera LL(k)



Nierekurencyjny analizator top-down



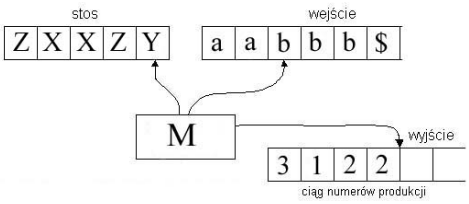
Konfiguracja parsera LL



Konfiguracja: (ZXXZY, bbb$, 3122)

stos | nieprzeczytana część wejścia | wyjście

Parser LL(k)



G=<V, Σ, P, S> – gramatyka opisująca składnię

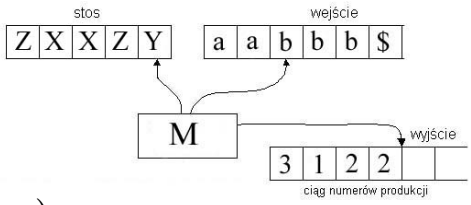
A = <Σ, Γ, Z_0, Δ, M, $>

- Σ – zbiór symboli terminalnych
- Γ – zbiór symboli stosowych
  - $\Gamma = V \cup \Sigma$
- $Z_0 = S$ – symbol początkowy stosu oraz gramatyki
- Δ – alfabet wyjściowy: zbiór numerów produkcji
- M – tablica sterująca parsera

$$M : V \times \left(\Sigma \cup \{\$\}\right)^* \mapsto \begin{cases} \left(V \cup \Sigma\right)^* \times \Delta \\ \underline{ERROR} \end{cases}$$

- $ - ogranicznik końca słowa wejściowego

Parser LL(k)



Konfiguracja parsera: $(\alpha Y, x, \pi)$

αY - zawartość stosu
Y - wierzchołek stosu
x - nieprzeczytana część wejścia
π - łańcuch numerów produkcji na wyjściu

Parser podgląda na wejściu $u = FIRST_K(x)$

Działanie parsera LL(k)

- Automat widzi wierzchołek stosu i *k* symboli wejściowych.
- Tablicę *M* wykorzystuje się, gdy na wierzchołku stosu znajduje się nieterminal.
- Konfiguracja początkowa: *(S, ω$, ε)* , przy czym:
  - *ω* – analizowane słowo;
  - *S* – symbol początkowy gramatyki

Działanie parsera LL(k)

Konfiguracja parsera: $(\alpha Y, x, \pi)$ zaś $u = FIRST_k(x)$

(1) if $Y \in \Sigma$ then
  if $Y = a$ then /*pop*/
    $(\alpha a, a\gamma, \pi) \mapsto (\alpha, \gamma, \pi)$
  else ERROR;

(2) if $Y \in V$ then
  if $M(Y,u) = (\beta, i)$ then /* wyprowadzenie */
    $(\alpha Y, x, \pi) \mapsto (\alpha\beta^R, x, \pi i)$
    /* $Y \to \beta$ jest produkcją o numerze i */
  else ERROR;

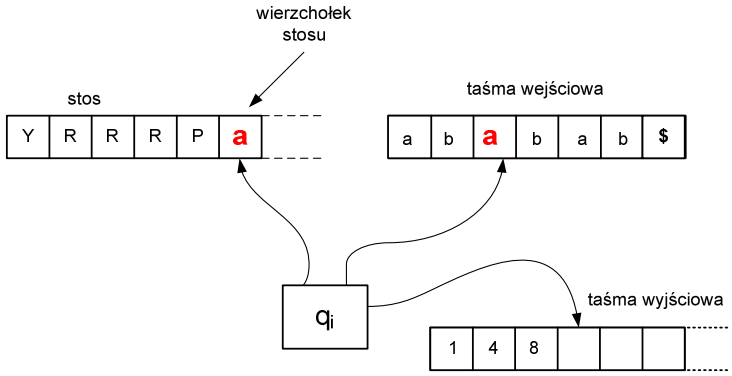
(3) if $(Y = \varepsilon)$ then
  if $x = \$$ then ACCEPT
  else ERROR;

Działanie parsera LL(k)

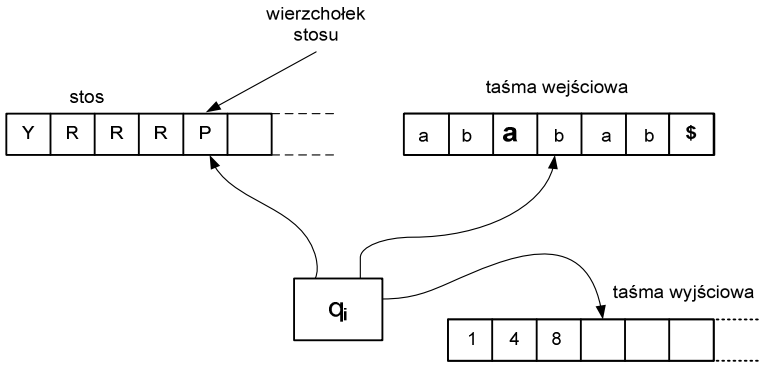
Konfiguracja końcowa akceptująca: $(\varepsilon, \$, \pi)$

π - ciąg numerów produkcji opisujący rozkład lewostronny słowa $\omega \in T^*$ w gramatyce G. Wówczas $\omega \in L(G)$

Krok „pop”

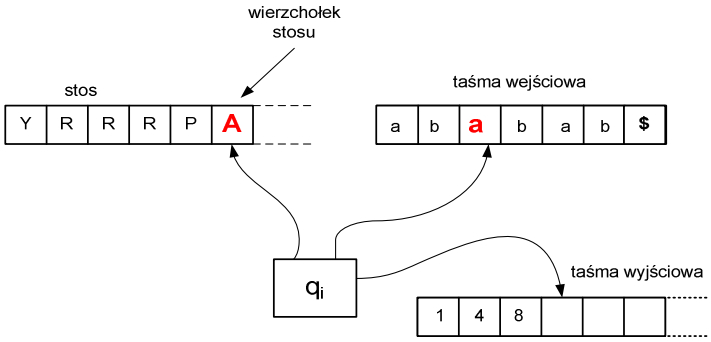


Krok „pop”



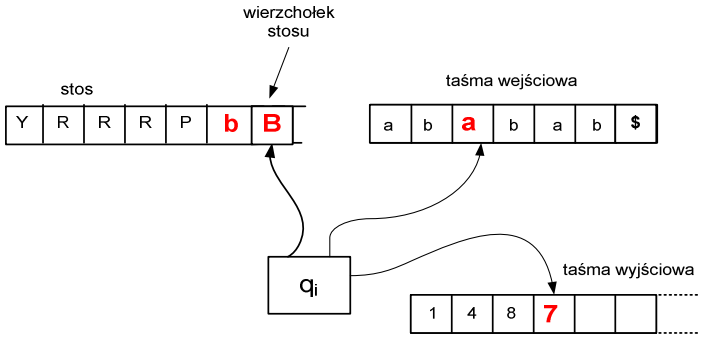
Krok „wyprowadzenia”

A→Bb



Krok „wyprowadzenia”

A→Bb



Konstrukcja tablicy sterującej M

Konstrukcja tablicy M dla parsera LL(1)

We: $G = \langle N, T, P, Z \rangle \in G_{BK}$

G – jest klasy LL(1)

Wy: tablica $M : V \times (\Sigma \cup \{\$\}) \mapsto \begin{cases} (V \cup \Sigma)^* \times \Delta \\ \underline{ERROR} \end{cases}$

gdzie: Δ - zbiór numerów produkcji w gramatyce G

Konstrukcja tablicy sterującej M

for $(A \to \beta) \in P$ and $A \to \beta$ - produkcja numer i do

begin

(1) for każde $a \in FIRST_1(\beta) : a \neq \varepsilon$ do

$M(A,a) := (\beta, i)$;

(2) if $\varepsilon \in FIRST_1(\beta)$ then

for każde $b \in FOLLOW_1(A)$ do

$M(A,b) := (\beta, i)$;

end;

for każde niezdefiniowane $M(A,a)$ do

$M(A,a) := \underline{ERROR}$;

Projektowanie tablicy parsera LL

Mając przekształconą gramatykę języka oraz wyznaczone (dla symboli nieterminalnych tej gramatyki) zbiory $FIRST_1$ i $FOLLOW_1$ możemy przystąpić do projektowania tablicy parsera LL.

1) $E \rightarrow TE'$
2) $E' \rightarrow +TE'$
3) $E' \rightarrow \varepsilon$
4) $T \rightarrow FT'$
5) $T' \rightarrow *FT'$
6) $T' \rightarrow \varepsilon$
7) $F \rightarrow (E)$
8) $F \rightarrow \underline{id}$

Tablica parsera

	FIRST	FOLLOW	id	+	*	(	)	$
E	(, id	$,)						
E'	+, ε	$,)						
T	(, id	+, $,)						
T'	*, ε	+, $,)						
F	(, id	*, +, $,)						

1) $E \rightarrow TE'$
2) $E' \rightarrow +TE'$
3) $E' \rightarrow \varepsilon$
4) $T \rightarrow FT'$
5) $T' \rightarrow *FT'$
6) $T' \rightarrow \varepsilon$
7) $F \rightarrow (E)$
8) $F \rightarrow \underline{id}$

Tablica parsera

	FIRST	FOLLOW	id	+	*	(	)	$
E	(, id	$,)	E→TE' (1)			E→TE' (1)		
E'	+, ε	$,)						
T	(, id	+, $,)						
T'	*, ε	+, $,)						
F	(, id	*, +, $,)						

1) E → TE′
2) E′ → +TE′
3) E′ → ε
4) T → FT′
5) T′ → *FT′
6) T′ → ε
7) F → (E)
8) F → id

Tablica parsera

	FIRST	FOLLOW	id	+	*	(	)	$
E	(, id	$,)	E→TE' (1)			E→TE' (1)		
E'	+, ε	$,)						
T	(, id	+, $,)	T→FT' (4)			T→FT' (4)		
T'	*, ε	+, $,)						
F	(, id	*, +, $,)						

1) E → TE′
2) E′ → +TE′
3) E′ → ε
4) T → FT′
5) T′ → *FT′
6) T′ → ε
7) F → (E)
8) F → id

Tablica parsera

	FIRST	FOLLOW	id	+	*	(	)	$
E	(, id	$,)	E→TE' (1)			E→TE' (1)		
E'	+, ε	$,)						
T	(, id	+, $,)	T→FT' (4)			T→FT' (4)		
T'	*, ε	+, $,)						
F	(, id	*, +, $,)	F→id (8)			F→(E) (7)		

1) E → TE′
2) E′ → +TE′
3) E′ → ε
4) T → FT′
5) T′ → *FT′
6) T′ → ε
7) F → (E)
8) F → id

Tablica parsera

	FIRST	FOLLOW	id	+	*	(	)	$
E	(, id	$,)	E→TE' (1)			E→TE' (1)		
E'	+, ε	$,)						
T	(, id	+, $,)	T→FT' (4)			T→FT' (4)		
T'	*, ε	+, $,)			T'→*FT' (5)			
F	(, id	*, +, $,)	F→id (8)			F→(E) (7)		

1) E → TE′
2) E′ → +TE′
3) E′ → ε
4) T → FT′
5) T′ → *FT′
6) T′ → ε
7) F → (E)
8) F → id

Tablica parsera

	FIRST	FOLLOW	id	+	*	(	)	$
E	(, id	$,)	E→TE' (1)			E→TE' (1)		
E'	+, ε	$,)						
T	(, id	+, $,)	T→FT' (4)			T→FT' (4)		
T'	*, ε	+, $,)		T'→ε (6)	T'→*FT' (5)		T'→ε (6)	T'→ε (6)
F	(, id	*, +, $,)	F→id (8)			F→(E) (7)		

Tablica parsera

	FIRST	FOLLOW	id	+	*	(	)	$
E	(, id	$,)	E→TE' (1)					
E'	+, ε	$,)		E'→+TE' (2)			E'→ε (3)	E'→ε (3)
T	(, id	+, $,)	T→FT' (4)					
T'	*, ε	+, $,)		T'→ε (6)	T'→*FT' (5)		T'→ε (6)	T'→ε (6)
F	(, id	*, +, $,)	F→id (8)			F→(E) (7)		

1) E → TE′
2) E′ → +TE′
3) E′ → ε
4) T → FT′
5) T′ → *FT′
6) T′ → ε
7) F → (E)
8) F → id

Tablica parsera

	FIRST	FOLLOW	id	+	*	(	)	$
E	(, id	$,)	E→TE' (1)			E→TE' (1)		
E'	+, ε	$,)		E'→+TE' (2)			E'→ε (3)	E'→ε (3)
T	(, id	+, $,)	T→FT' (4)			T→FT' (4)		
T'	*, ε	+, $,)		T'→ε (6)	T'→*FT' (5)		T'→ε (6)	T'→ε (6)
F	(, id	*, +, $,)	F→id (8)			F→(E) (7)		

Symulacja działania parsera LL

Stos	**Wejście**	**Wyjście**
E	id+id$	ε
E'T	id+id$	1
E'T'F	id+id$	14
E'T'id	id+id$	148
E'T'	+id$	148
E'	+id$	1486
E'T+	+id$	14862
E'T	id$	14862
E'T'F	id$	148624
E'T'id	id$	1486248
E'T'	$	1486248
E'	$	14862486
ε	$	148624863

akceptacja

Metoda zejść rekurencyjnych

(1) Dla każdego symbolu nieterminalnego gramatyki budujemy procedurę (funkcję) rekurencyjną. Z programu nadrzędnego (głównego) wywoływana jest procedura (funkcja) dla symbolu początkowego gramatyki.

(2) Wewnątrz procedury (funkcji) dokonuje się wyboru produkcji na podstawie tokenu podglądanego na wejściu (lookahead). Po dokonaniu wyboru zapewnia się zanotowanie numeru wybranej produkcji (raport). Następnie każdy symbol nieterminalny prawej strony produkcji przekształca się w wywołanie odpowiadającej mu procedury (funkcji), każdy symbol terminalny prowadzi do wywołania operacji „match", która sprawdza obecność tego symbolu na wejściu. W przypadku zgodności czyta następny token, w przeciwnym razie sygnalizuje błąd. Symbole z prawej strony produkcji znajdują swoje odpowiedniki w procedurze w tej kolejności, w jakiej są one umieszczone w prawej stronie produkcji. Niemożność dokonania wyboru produkcji wewnątrz procedury sygnalizowana jest błędem.

Metoda zejść rekurencyjnych

	id	+	*	(	)	$
E	(1)$E \to TE_1$			(1)$E \to TE_1$		
E_1		(2)$E_1 \to +TE_1$			(3)$E_1 \to \varepsilon$	(3)$E_1 \to \varepsilon$
T	(4)$T \to FT_1$			(4)$T \to FT_1$		
T_1		(6)$T_1 \to \varepsilon$	(5)$T_1 \to *FT_1$		(6)$T_1 \to \varepsilon$	(6)$T_1 \to \varepsilon$
F	(8)$F \to \underline{id}$			(7)$F \to (E)$		

```
E()
{
   if (lookahead == ID || lookahead == '(')
   {
        raport(1); T(); E1();
   }
   else error();
}
```

Metoda zejść rekurencyjnych

	id	+	*	(	)	$
E	$(1)E \to TE_1$			$(1)E \to TE_1$		
E_1		$(2)E_1 \to +TE_1$			$(3)E_1 \to \varepsilon$	$(3)E_1 \to \varepsilon$
T	$(4)T \to FT_1$			$(4)T \to FT_1$		
T_1		$(6)T_1 \to \varepsilon$	$(5)T_1 \to *FT_1$		$(6)T_1 \to \varepsilon$	$(6)T_1 \to \varepsilon$
F	$(8)F \to \underline{id}$			$(7)F \to (E)$		

```
E1()
{
   if(lookahead == '+')
   {
        raport(2); match('+'); T(); E1();
   }
   else if (lookahead == ')' || lookahead == '$')
        raport(3);
   else error();
}
```

Metoda zejść rekurencyjnych

	id	+	*	(	)	$
E	$(1)E \to TE_1$			$(1)E \to TE_1$		
E_1		$(2)E_1 \to +TE_1$			$(3)E_1 \to \varepsilon$	$(3)E_1 \to \varepsilon$
T	$(4)T \to FT_1$			$(4)T \to FT_1$		
T_1		$(6)T_1 \to \varepsilon$	$(5)T_1 \to *FT_1$		$(6)T_1 \to \varepsilon$	$(6)T_1 \to \varepsilon$
F	$(8)F \to \underline{id}$			$(7)F \to (E)$		

```
T()
{
   if(lookahead == ID || lookahead == '(')
   {
        raport(4); F(); T1();
   }
   else error();
}
```

Metoda zejść rekurencyjnych

	id	+	*	(	)	$
E	$(1)E \to TE_1$			$(1)E \to TE_1$		
E_1		$(2)E_1 \to +TE_1$			$(3)E_1 \to \varepsilon$	$(3)E_1 \to \varepsilon$
T	$(4)T \to FT_1$			$(4)T \to FT_1$		
T_1		$(6)T_1 \to \varepsilon$	$(5)T_1 \to *FT_1$		$(6)T_1 \to \varepsilon$	$(6)T_1 \to \varepsilon$
F	$(8)F \to \underline{id}$			$(7)F \to (E)$		

```
T1()
{
   if (lookahead == '*')
   {
        raport(5); match('*'); F(); T1();
   }
   else if(lookahead == '+' || lookahead == ')' ||
         lookahead == '$')  raport(6);
   else error();
}
```

Metoda zejść rekurencyjnych

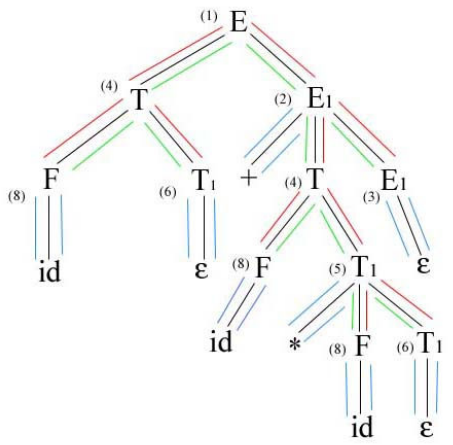
	id	+	*	(	)	$
E	$(1)E \to TE_1$			$(1)E \to TE_1$		
E_1		$(2)E_1 \to +TE_1$			$(3)E_1 \to \varepsilon$	$(3)E_1 \to \varepsilon$
T	$(4)T \to FT_1$			$(4)T \to FT_1$		
T_1		$(6)T_1 \to \varepsilon$	$(5)T_1 \to *FT_1$		$(6)T_1 \to \varepsilon$	$(6)T_1 \to \varepsilon$
F	$(8)F \to \underline{id}$			$(7)F \to (E)$		

```
F()
{
   if(lookahead == ID)
   {
        raport(8); match(ID);
   }
   else if(lookahead == '(')
   {
        raport(7); match('('); E(); match(')');
   }
   else error();
}
```

Metoda zejść rekurencyjnych

```
error()
{ ....... }

match(t)
int t;
{
   if(lookahead == t)  lookahead=gettoken();
   else error();
}

raport()
{ ....... }

gettoken()
{ ....... }

int lookahead;
main()
{
   lookahead=gettoken();
   E();            /*wywołanie procedury dla symbolu
                     początkowego gramatyki*/
}
```

Metoda zejść rekurencyjnych



wywołania (zejścia) rekurencyjne

powroty

działanie procedur nie powodujących zejść rekurencyjnych

(i) raportowanie numerów stosowanych produkcji

Wejście:
id + id * id