Преобразуване БНФ – синтактичен – граф – програма за синтактичен анализ. Таблично управляем анализ

Много често работата на синтактичния анализатор се представя чрез т.нар. синтактичен граф.

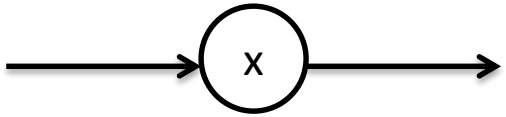
Отличителна черта на анализа отгоре-надолу (top-down) е, че целта на анализа е предварително известна – разпознаване на изречение (символен низ), което може да бъде породено (генерирано) от стартовия символ. Прилагането на дадено правило съответства на разделяне на основната задача на множество от подзадачи.

Програмната реализация съответства на нетерминалните символи и съответните подзадачи.

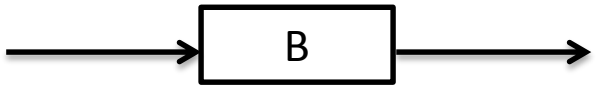
Неоходимо е да построим граф, който представя цялата програма за целите на синтактичния анализ.

Построяване на синтактичен граф

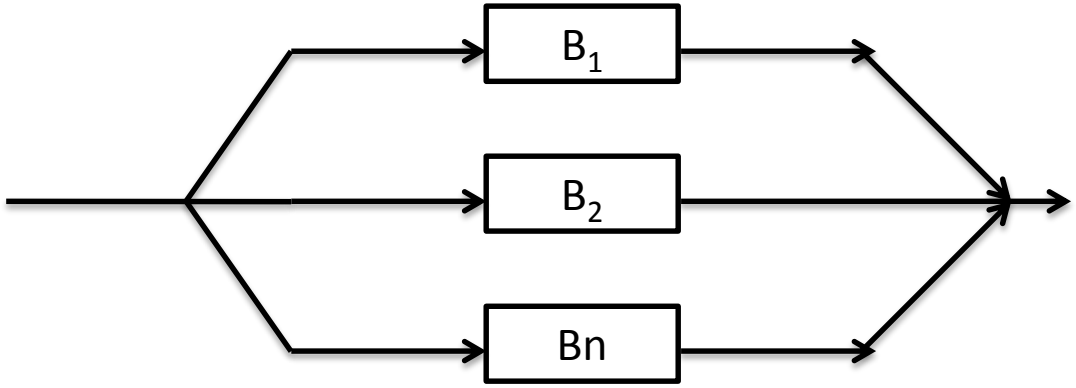
1. Терминален символ



2. Нетерминален символ

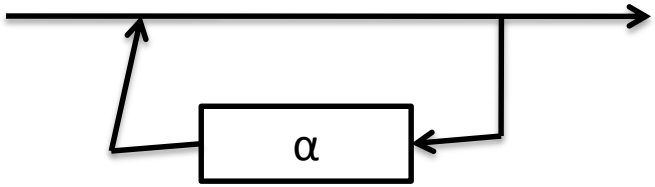


3. Правило от вида A::= B_1 |B_2 | … | B_n



4. Правило от вида $A ::= B_1 B_2 \ldots B_n$

5. Правило от вида $B ::= \{\alpha\}$



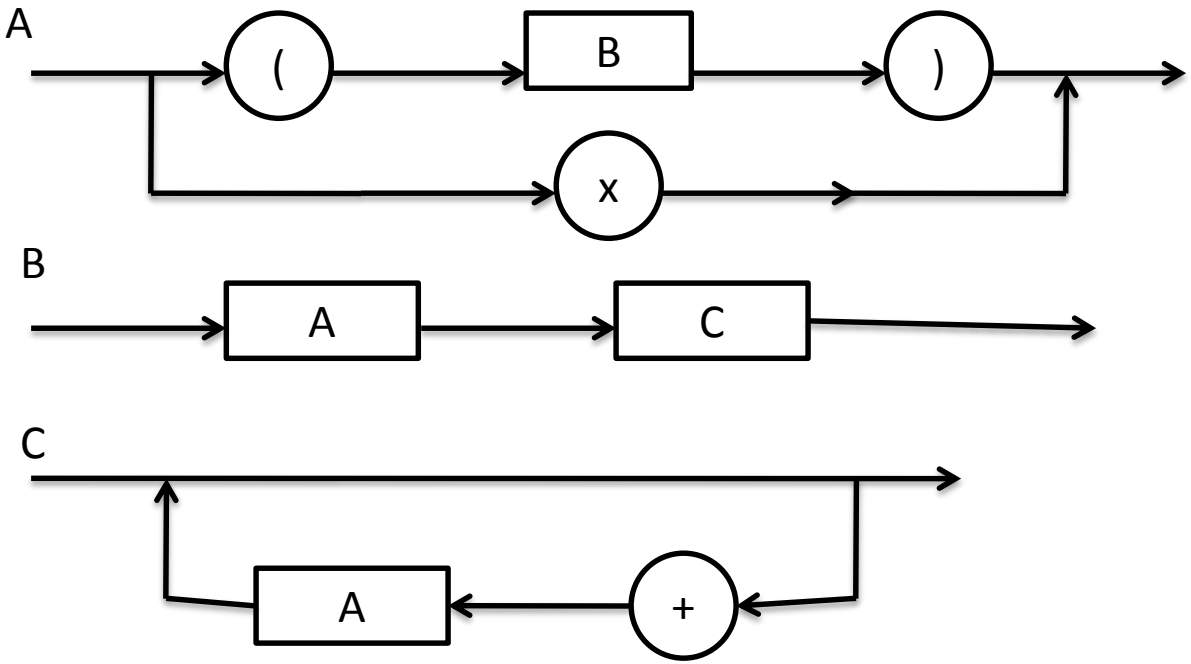
Пример

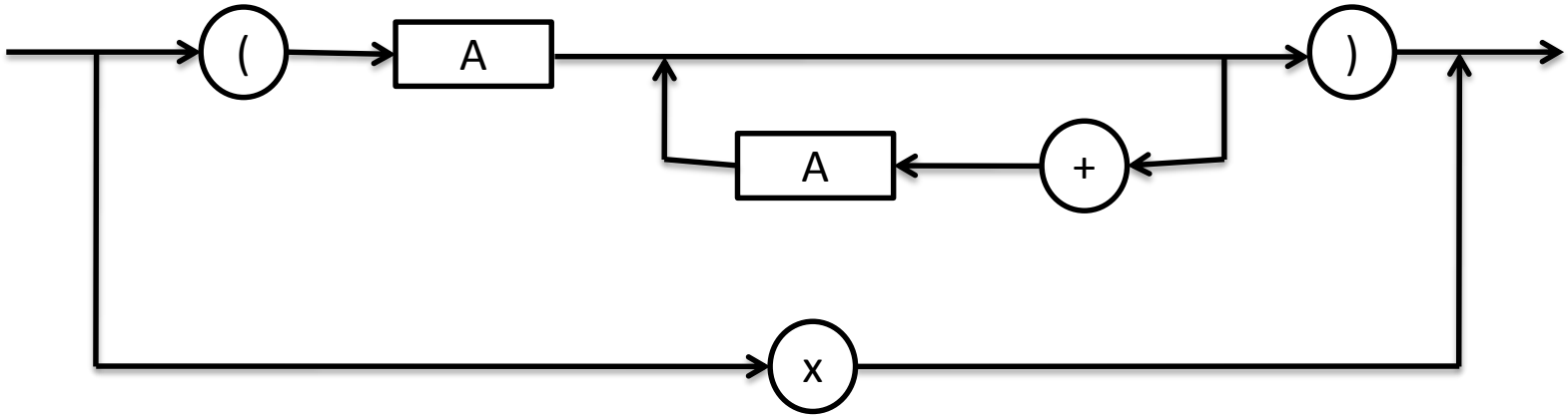
A ::= x | (B)

B ::= AC

C ::= {+A}

поражда x, (x), (x+x), ((x))




(
A
)
A
+
x

Преобразуване на граф в програма

1. Последователно

```
{
        P(S1);
        P(S2);
        ….
        P(Sn);
}
```

2. Разклонение

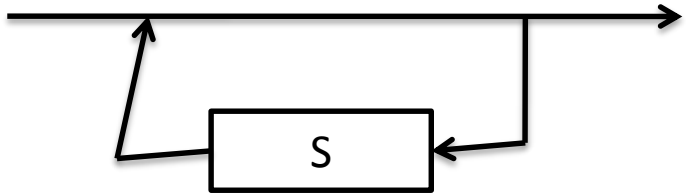
A.

```
if  (ch IN L1)  P(S1);
else  if ( ch IN L2) P(S2);
        . . .
else if (ch IN Ln ) P(Sn);
else error();
```

B.

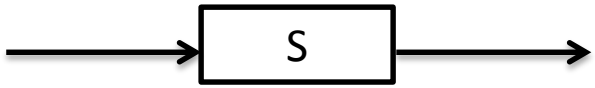
```
switch (ch) {
      case  L1: P(S1); break
      case  L2: P(S2); break;
       . . .
      case  Ln: P(Sn); break;
      default:    error();; break;
}
```

Where Li = first (Si)



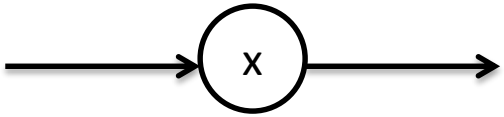
3. Повторение

```
while (ch IN L) { P(S); }
```



4. Нетерминал

```
P(S);
```



5. Терминал

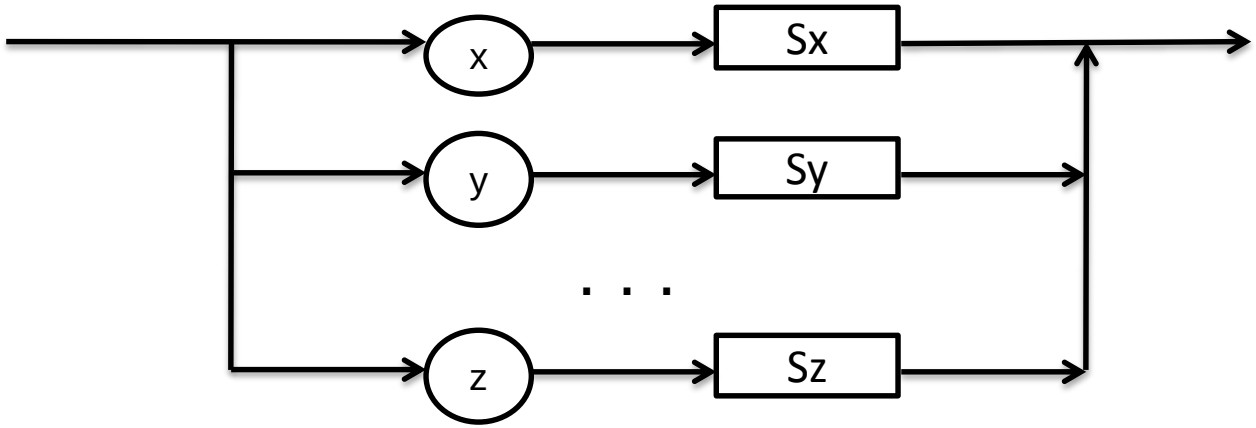
```
if (ch == x)
    ch = getchar ();
else
    error ();
```

Програмна реализация на анализ

```
# include <stdio.h>
extern “C” void error ();
void B();
void C();
char ch;
void A() {
      if (ch == ‘(‘ ) {
            ch = getchar ();
            B ();
             if ( ch == ‘)’ ) ch = getchar ();
             else error ();
      } else
              error ();
}
```

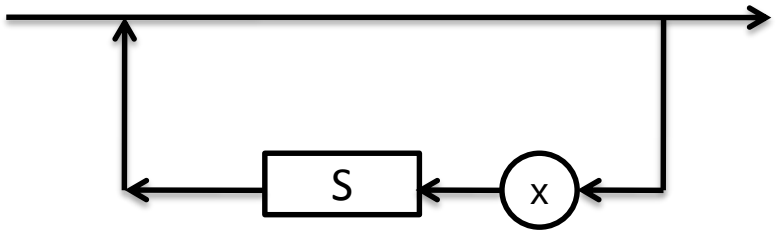
```
void B () {
        A ();
        C ();
}
void C () {
        while ( ch == ‘+’ ) {
                ch = getchar ();
                A ();
        }
}
void main () {
        ch = getchar ();
        A();
}
```

6. Разклонение с допълнителни нетерминални символи




```
if (ch == ‘x’ ) {ch = getchar(); P(Sx); }
else  if (ch == ‘y’) { ch = getchar (); P(Sy); }
. . .
 else if (ch == ‘z’ ) { ch = getchar (); P(Sz);
 else  error();
```

Повторение с допълнение




```
while (ch == ‘x’ ) {
        ch = getchar ();
         P(S);
}
```

8. Заместване на “while” с “do while”

```
ch = getchar();
P(S);
while (ch == ‘x’ ) {
        ch =  getchar ();
         P(S);
}

do {
        ch = getchar ();
         P (S);
} while (ch == ‘x’);
```

Таблично управляван анализатор

- Построява се таблица T за граматика G
- За всяко правило A -> α от G:
  - За всеки терминал t от First(α)
    - T[A, t] = α
  - Ако ε е от First(α) за всяко t от Follow(A)
    - T[A, t] = α
  - Ако ε е от First(α) и $ е от Follow(A)
    - T[A, $] = α

Таблица

E -> TX

T -> (E) | intY

X -> +E | ε

Y -> *T | ε

Follow(X) = {$,)}

Follow(Y) = {+,$,)}

	(	)	+	*	int	$
E	TX				TX	
T	(E)				intY	
X	Error	ε	+E			ε
Y		ε	ε	*T		ε

Проблем

- Пример:

S -> Sa | b

First(S) = {b}

Follow(S) = {$,a}

	a	b	$
S		**b** **Sa**	

Извод

- Ако в таблицата има колона с два записа, то граматиката G не е LL(1), т.е. е в сила едно от следните твърдения:
  - Не е с лява факторизация
  - Има лява рекурсия
  - Неоднозначна
- Граматиките на много програмни езици са контекстно-свободни, но не са LL(1)