

10. Синтаксический анализ. Часть 2

Разделы:

- s - и q -грамматики
- $LL(1)$ -грамматики
- Нерекursивный алгоритм анализа $LL(1)$ -грамматик
- Метод рекурсивного спуска
- $LL(k)$ -грамматики с $k > 1$

S-грамматики

JFLAP : <untitled1>

File Input Test Convert Help

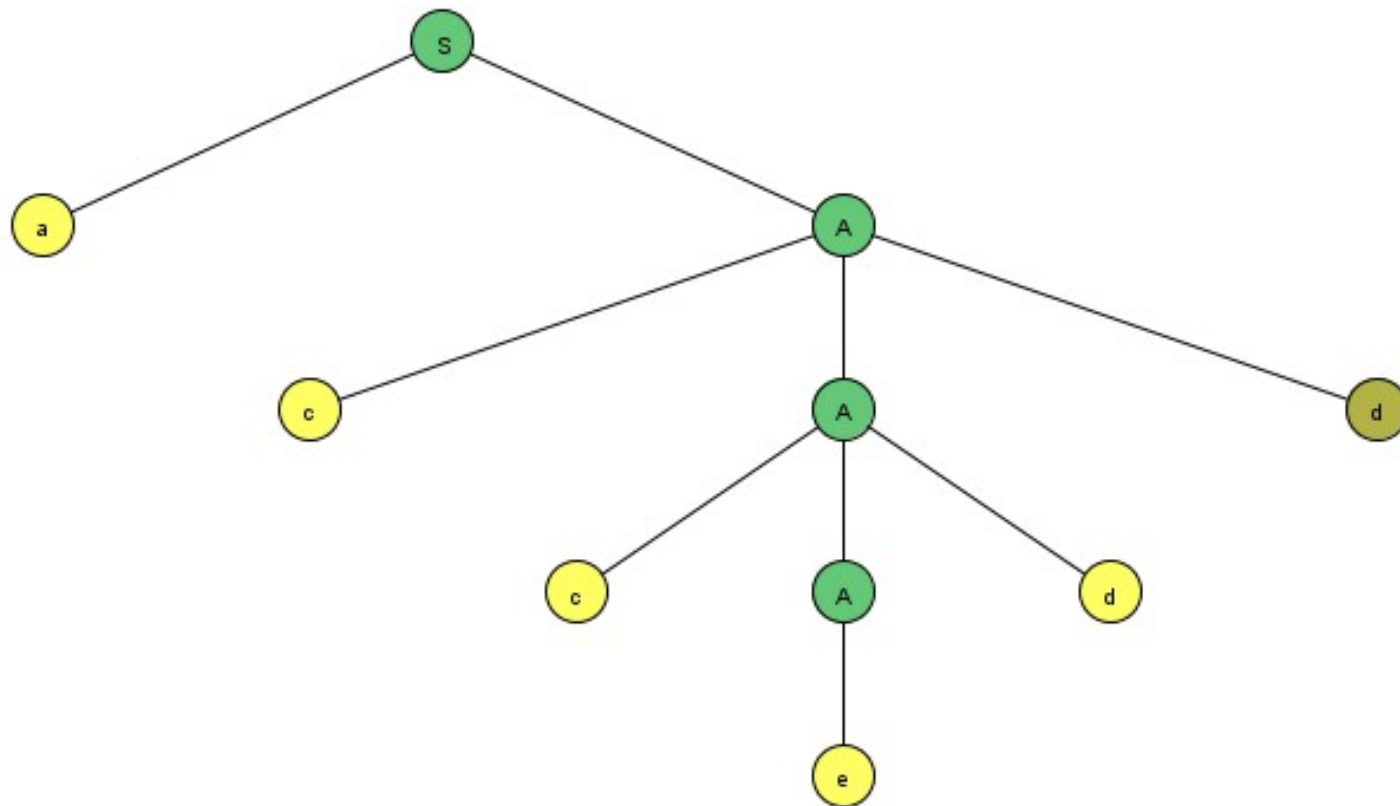
Editor

Table Text Size

LHS		RHS
S	→	aA
S	→	bB
A	→	cAd
A	→	e
B	→	cBf
B	→	g

S-грамматики

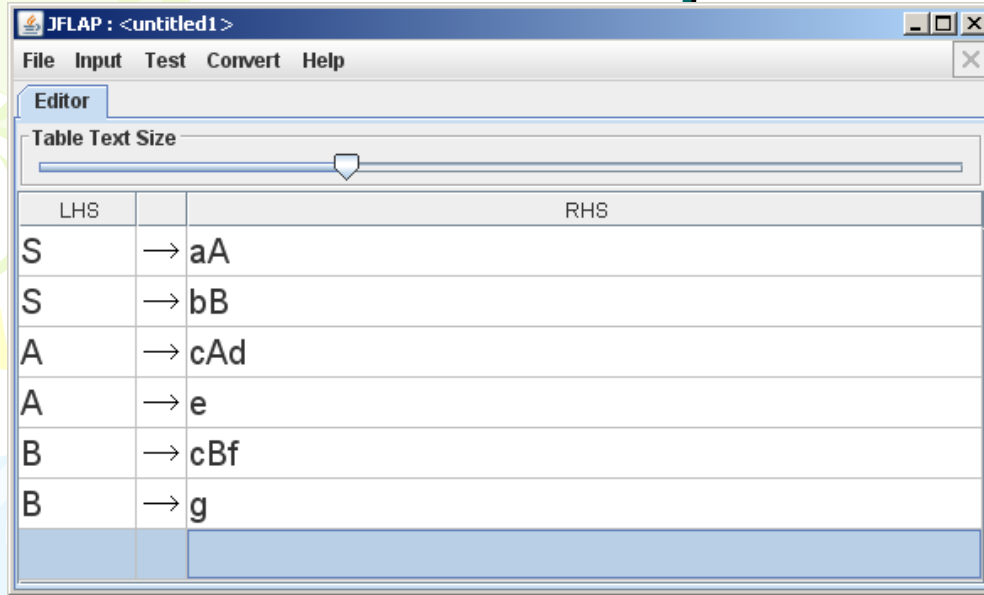
- Входная строка *accedd*
- Дерево разбора ниже на схеме



S-грамматики

- В каждой продукции RHS начинается с терминального символа
- Альтернативы начинаются с различных символов
- Строится **детерминированный синтаксический анализатор**
- Можно на основе МПА
 - **Конфигурация** анализатора - $(ax, \alpha\perp, \pi)$
 - ax – это необработанная часть входной строки, a – текущий входной символ, $\alpha\perp$ – содержимое магазина, π – разбор
 - Разбор – цепочка номеров применяемых на шаге порождения продукции

S-грамматики



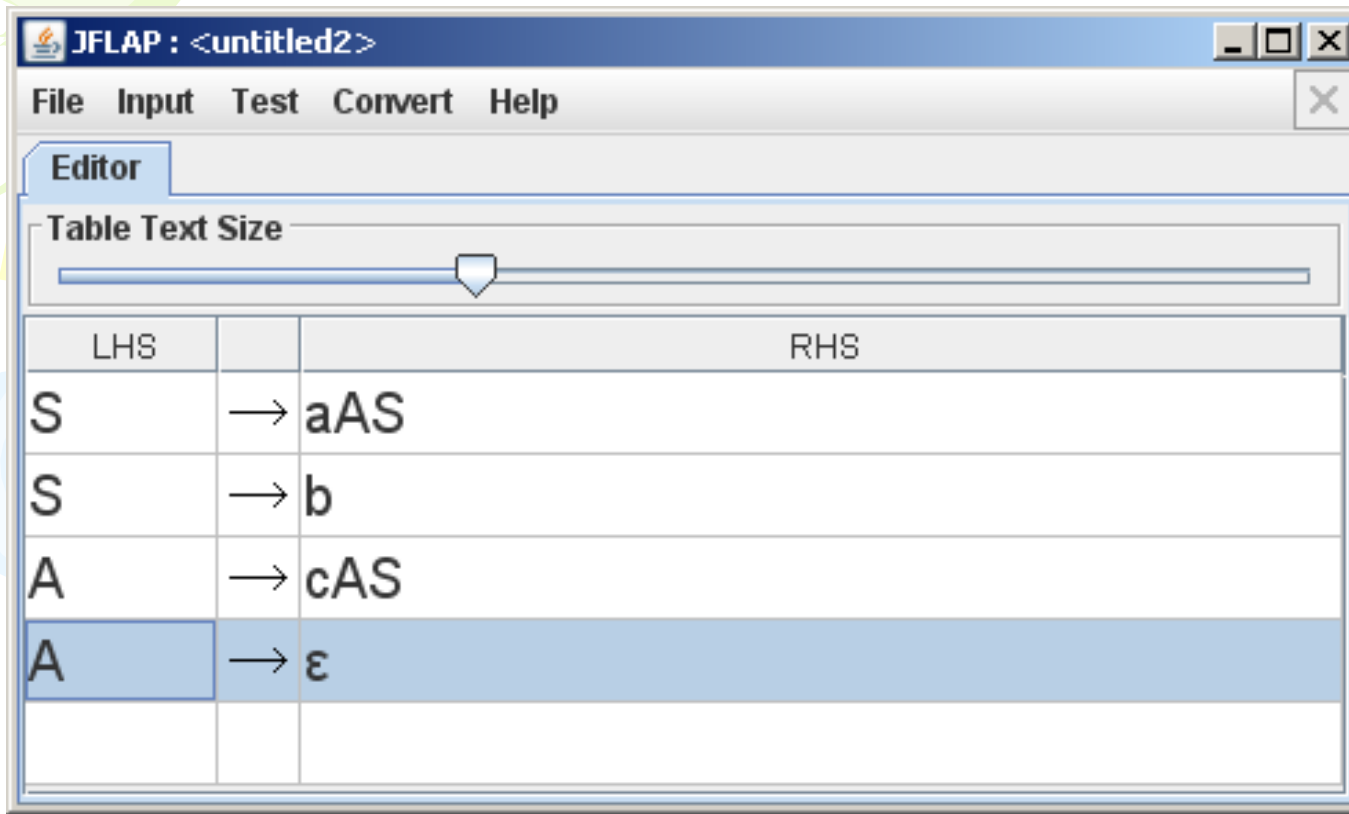
- Это простая $LL(1)$ -грамматика, или **s-грамматика**

	Входная строка		Магазин	Действия анализатора
1		<i>accedd</i>	$S\perp$	Применить первую продукцию
2		<i>accedd</i>	$aA\perp$	Вытолкнуть
3	<i>a</i>	<i>ccedd</i>	$A\perp$	Применить третью продукцию
4	<i>a</i>	<i>ccedd</i>	$cAd\perp$	Вытолкнуть
5	<i>ac</i>	<i>cedd</i>	$Ad\perp$	Применить третью продукцию
6	<i>ac</i>	<i>cedd</i>	$cAdd\perp$	Вытолкнуть
7	<i>acc</i>	<i>edd</i>	$Add\perp$	Применить четвертую продукцию
8	<i>acc</i>	<i>edd</i>	$edd\perp$	Вытолкнуть
9	<i>acce</i>	<i>dd</i>	$dd\perp$	Вытолкнуть
10	<i>acced</i>	<i>d</i>	$d\perp$	Вытолкнуть
11	<i>accedd</i>	ϵ	\perp	Допустить

S-грамматики

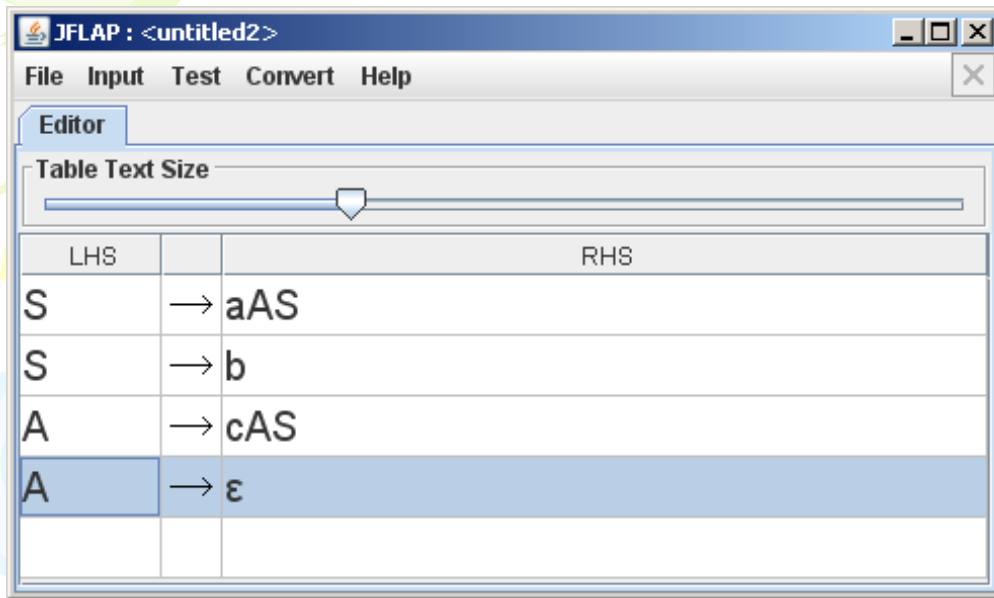
- КСГ $G = (V_N, V_T, P, S)$ без ε -продукций - **простая** LL(1)-грамматика, если для каждого нетерминала A все альтернативные RHS начинаются с разных терминальных символов
- Можно сформулировать правила построения ДМПА, выполняющего НСА
 - 1) замена верхнего элемента магазина на строку, которая является RHS некоторой продукции;
 - 2) выталкивание элемента из магазина
- «Проблемка»: большинство конструкций ЯП не могут быть описаны с их помощью

q-грамматики



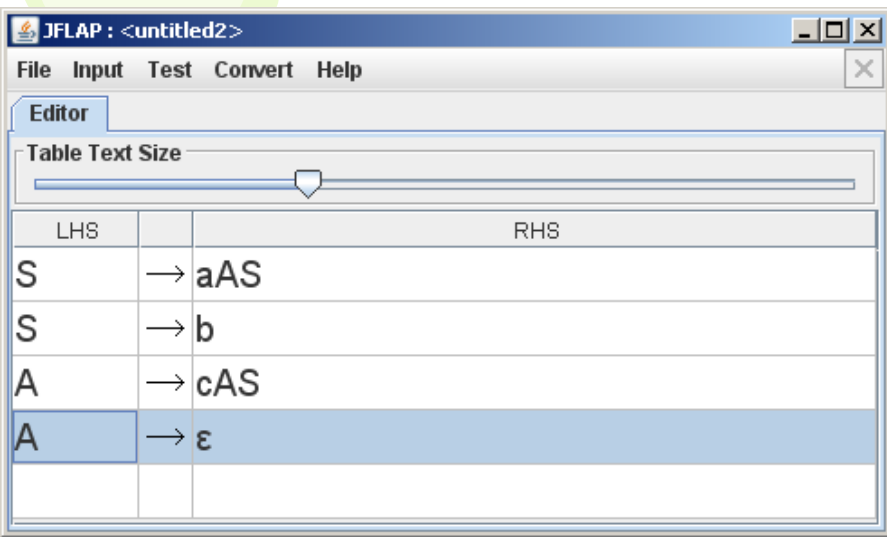
• **СТУДЕНТАМ:** почему данная грамматика не является s-грамматикой?

q-грамматики



	Входная строка		Магазин	Действия анализатора
1		<i>acbb</i>	$S\perp$	Применить первую продукцию
2		<i>acbb</i>	$aAS\perp$	Вытолкнуть
3	<i>a</i>	<i>cbb</i>	$AS\perp$	Применить третью продукцию
4	<i>a</i>	<i>cbb</i>	$cASS\perp$	Вытолкнуть
5	<i>ac</i>	<i>bb</i>	$ASS\perp$	Применить четвертую продукцию
6	<i>ac</i>	<i>bb</i>	$SS\perp$	Применить вторую продукцию
7	<i>ac</i>	<i>bb</i>	$bS\perp$	Вытолкнуть
8	<i>acb</i>	<i>b</i>	$S\perp$	Применить вторую продукцию
9	<i>acb</i>	<i>b</i>	$b\perp$	Вытолкнуть
10	<i>acbb</i>	ϵ	\perp	Допустить

q-грамматики



The image shows the JFLAP software interface. The title bar reads 'JFLAP : <untitled2>'. The menu bar includes 'File', 'Input', 'Test', 'Convert', and 'Help'. Below the menu is an 'Editor' tab. A 'Table Text Size' slider is visible. The main area contains a table with two columns: 'LHS' and 'RHS'. The table has four rows of grammar rules. The fourth row, 'A → ε', is highlighted in blue.

LHS	RHS
S	→ aAS
S	→ b
A	→ cAS
A	→ ε

- $FIRST(A) = \{a \text{ из } V_T \mid A \Rightarrow^+ \alpha\beta, \text{ где } \beta \text{ принадлежит } (V_N \cup V_T)^*\}$
- $FOLLOW(A) = \{a \text{ из } V_T \mid S \Rightarrow^* \alpha A \gamma, \text{ и } a \text{ принадлежит } FIRST(\gamma)\}$

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

$$FOLLOW(S) = \{\perp\}$$

$$FIRST(A) = \{c\}$$

$$FOLLOW(A) = FIRST(S) = \{a, b\}$$

q-грамматики

- При СА применяется некоторая продукция, если анализатор находится в конфигурации $(ax, Aa\perp, \pi)$ и выполняется одно из условий:
 - продукция имеет вид $A \rightarrow a\beta$;
 - продукция имеет вид $A \rightarrow \varepsilon$ и символ a принадлежит $FOLLOW(A)$
- Множество выбора продукций (множество-селектор) *SELECT*, которое определяется для продукции следующим образом:
 - если продукция имеет вид $A \rightarrow a\beta$, то $SELECT(A \rightarrow a\alpha) = FIRST(a\alpha) = \{a\}$;
 - если продукция имеет вид $A \rightarrow \varepsilon$, то $SELECT(A \rightarrow a\alpha) = FOLLOW(A)$

q -грамматики

- КСГ $G = (VN, VT, P, S)$ - **q -грамматика**, если RHS начинается с терминала или пуста, и множества-селекторы для каждого нетерминала A **не пересекаются**
- Построение МПА по q -грамматике может быть выполнено так же, как и по s -грамматике
- Это более мощный класс по сравнению с s -грамматиками

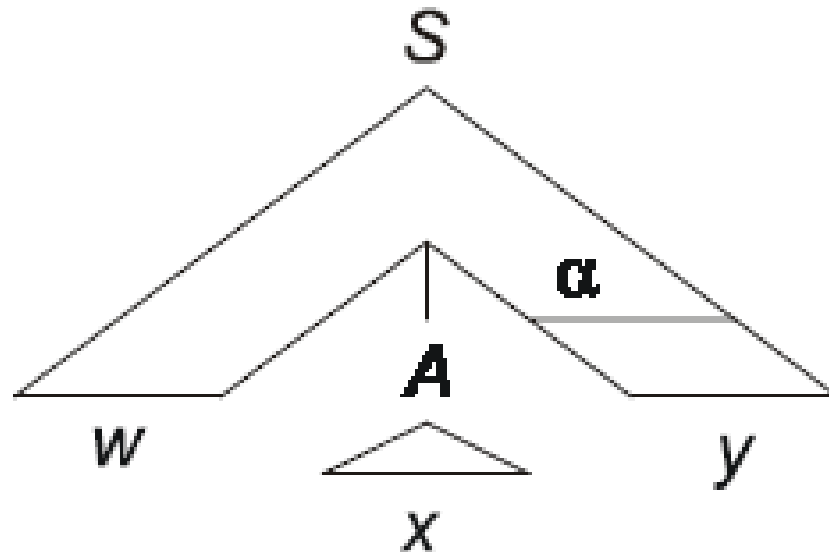
LL(1)-грамматики

- Более общее определение *FIRST*-множества
- $FIRST(\alpha) = \{a \text{ из } V_T \mid S \Rightarrow^* \alpha \Rightarrow^* a\beta, \text{ где } \alpha \text{ принадлежит } (V_N \cup V_T)^+, \beta \text{ принадлежит } (V_N \cup V_T)^*\}$
- КСГ G - **LL(1)-грамматика**, если из:
 - 1) $S \Rightarrow^* wA\alpha \Rightarrow^* w\beta\alpha \Rightarrow^* wx$
 - 2) $S \Rightarrow^* wA\alpha \Rightarrow^* w\gamma\alpha \Rightarrow^* wy$, для которых из $FIRST(x) = FIRST(y)$,
- следует, что $\beta = \gamma$
- «Проблемка»:
- согласно данному определению выбор продукции для замены A в $wA\alpha$ определяется w и следующим входным символом, но это не так

LL(1)-грамматики

- КСГ G является $LL(1)$ -грамматикой тогда и только тогда, когда для двух различных продукций $A \rightarrow \alpha$ и $A \rightarrow \beta$ из P пересечение $FIRST(\beta\alpha) \cap FIRST(\gamma\alpha) = \emptyset$ при всех таких $wA\alpha$, что $S \Rightarrow^* wA\alpha$
- Без доказательства
- Здесь для $LL(1)$ -грамматики продукция для замены нетерминала однозначно определяется текущим входным символом

LL(1)-грамматики



- G является LL(1) тогда и только тогда, когда для двух различных продукций $A \rightarrow \beta$ и $A \rightarrow \gamma$ из P справедливо $FIRST(\beta FOLLOW(A)) \cap FIRST(\gamma FOLLOW(A)) = \emptyset$ для всех A
- Без доказательства
- **СТУДЕНТАМ:** когда и какому множеству может принадлежать ε ?

LL(1)-грамматики

- Оба определения выше не являются конструктивными, а следующее – является
- G - LL(1) тогда и только тогда, когда для каждой A -продукции грамматики ($A \rightarrow a_1 \mid \dots \mid a_n, n > 0$) выполняются следующие условия:
 - Множества $FIRST(a_1), \dots, FIRST(a_n)$ попарно не пересекаются
 - Если $a_i \Rightarrow^* \varepsilon$, то $FIRST(a_j) \cap FOLLOW(A) = \emptyset$ для $1 \leq j \leq n, i \neq j$
- Следствие: леворекурсивная грамматика не может быть LL(1)-грамматикой

Нерекурсивный LL(1)-анализ

- Можно осуществить с помощью так называемого 1-предиктивного алгоритма, использующего TCA
- TCA $M[A, a]$ строится из FIRST- и FOLLOW-множеств и представляет собой двумерный массив, где A – нетерминал, а a – терминал или символ $\$$
- Если очередной входной символ a находится во множестве $FIRST(a)$, выбирается продукция $A \rightarrow a$.
- Когда $a \Rightarrow^* \varepsilon$ мы снова должны выбрать $A \rightarrow a$, если текущий входной символ имеется в $FOLLOW(A)$ или если из входного потока получен $\$$, который при этом входит в $FOLLOW(A)$

Нерекурсивный LL(1)-анализ

- Вход: грамматика G и множества первых порождаемых символов и символов-последователей
- Выход: ТСА $M[A, a]$
- Для каждой A -продукции ($A \rightarrow a$) в G выполняем действия:
 1. Для каждого терминала a из $FIRST(a)$ добавляем $A \rightarrow a$ в ячейку $M[A, a]$
 2. Если ϵ принадлежит $FIRST(a)$, то для каждого терминала b из $FOLLOW(A)$ добавляем $A \rightarrow a$ в $M[A, b]$. Если ϵ принадлежит $FIRST(a)$ и $\$$ принадлежит $FOLLOW(A)$, то добавляем $A \rightarrow a$ также и в $M[A, \$]$
 3. Если после выполнения этих действий ячейка $M[A, a]$ осталась без продукции, устанавливаем ее значение равным *error*

Нерекурсивный LL(1)-анализ

JFLAP : (14-ExpressionsCFG.jff)

File Input Test Convert Help

Editor Build LL(1) Parse

Do Selected Do Step Do All Next Parse

E	→	TA
A	→	+TA
A	→	ϵ
T	→	FB
B	→	*FB
B	→	ϵ
F	→	i
F	→	(E)

Parse table complete. Press "parse" to use it.

	FIRST	FOLLOW
A	{ ϵ , +}	{\$, , }
B	{ ϵ , *}	{\$, +, , }
E	{(, i}	{\$, , }
F	{(, i}	{\$, *, +, , }
T	{(, i}	{\$, +, , }

	()	*	+	i	\$
A		ϵ		+TA		ϵ
B		ϵ	*FB	ϵ		ϵ
E	TA				TA	
F	(E)				i	
T	FB				FB	

Нерекурсивный LL(1)-анализ

JFLAP : (14-IfElseCFG.jff)

File Input Test Convert Help

Editor Build LL(1) Parse

Do Selected Do Step Do All Next Parse

S	→	iEtSO
S	→	a
O	→	eS
O	→	ε
E	→	b

Parse table complete, but has ambiguity.

	FIRST	FOLLOW
E	{ b }	{ t }
O	{ ε, e }	{ e, \$ }
S	{ a, i }	{ e, \$ }

	a	b	e	i	t	\$
E		b				
O			ε eS			ε
S	a			iEtSO		

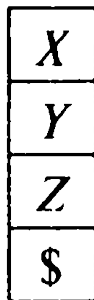
- LL(1)-грамматики не дают множественных записей ТСА

Нерекурсивный LL(1)-анализ

Входной буфер



Стек



Программа
предиктивного
синтаксического
анализа

Выходной поток

Таблица
синтаксического
анализа M

Нерекурсивный LL(1)-анализ

- Вход: строка w и таблица M для грамматики G
- Выход: если w принадлежит $L(G)$ – левое порождение w ; в противном случае – сообщение об ошибке
- Методика: изначально синтаксический анализатор находится в конфигурации с $w\$$ во входном буфере и аксиомой S грамматики G на вершине стека, над $\$$

Нерекурсивный LL(1)-анализ

- Использование ТСА M

Устанавливаем указатель входного потока ip так, чтобы он указывал на первый символ строки w ;

Устанавливаем X равным символу на вершине стека;

```
while (x <> $)/* Стек не пуст */
```

```
{
```

```
    Устанавливаем  $a$  равным символу, на который в настоящий момент указывает  $ip$   
    if ( $X == a$ )
```

```
    {
```

```
        Снимаем символ со стека и перемещаем  $ip$  к следующему символу строки;
```

```
    }
```

```
    else if ( $X$  — терминал) error();
```

```
    else if ( $M[X, a]$  — запись об ошибке ) error();
```

```
    else if ( $M[X, a]$  —  $X \rightarrow Y_1Y_2...Y_k$ )
```

```
    {
```

```
        Выводим продукцию  $X \rightarrow Y_1Y_2...Y_k$ ;
```

```
        Снимаем символ со стека;
```

```
        Помещаем в стек  $Y_k, Y_{k-1}, ..., Y_1$ ; /*  $Y_1$  помещается на вершину стека; */
```

```
    }
```

```
    Устанавливаем  $X$  равным символу на вершине стека;
```

```
}
```

Нерекурсивный LL(1)-анализ

- Псевдокод анализа методом рекурсивного спуска типичного нетерминала

```
void A()  
{  
    Выбираем A-продукцию  $A \rightarrow X_1X_2 \dots X_k$ ;  
    for (i от 1 до k)  
    {  
        if ( $X_i$  — нетерминал)  
            Вызов процедуры  $X_i()$ ;  
        else if ( $X_i$  равно текущему входному символу a)  
            Переходим к следующему входному символу;  
        else /* Обнаружена ошибка */  
            ;  
    }  
}
```

Метод рекурсивного спуска

PROGRAM \rightarrow begin DECLIST comma STMTLIST end

DECLIST \rightarrow d X

X \rightarrow semi DECLIST | ε

STMTLIST \rightarrow s Y

Y \rightarrow semi STMTLIST | ε

Метод рекурсивного спуска

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
/* Terminals */
#define begin 0
#define comma 1
#define end 2
#define semi 3
#define d 4
#define s 5
/* EOP */
#define EOP 6
#define UNDEF 7
/* Lexeme class */
int lexeme = 0;
/* Non terminals */
void PROGRAM(), DECLIST(), STMTLIST();
void X(), Y();
/* Reject input line */
void error();
/* Look ahead lexeme */
int get_token();
char g_inputBuffer[1024] = "";
char* g_prog = NULL;
```

Метод рекурсивного спуска

```
int main(int argc, char* argv[])
{
    if (1 == argc)
    {
        printf("Enter your input line: ");
        gets(g_inputBuffer);
    }
    else
    {
        strcpy(g_inputBuffer, argv[1]);
    }
    g_prog = (char*)g_inputBuffer;
    lexeme = get_token();
    PROGRAM();
    if (lexeme == EOP)
        printf("Accept.\n");
    else
        printf("Reject.\n");
}
```

Метод рекурсивного спуска

```
void PROGRAM()
{
    if (lexeme != begin)
        error();
    lexeme = get_token();
    DECLIST();
    if (lexeme != comma)
        error();
    lexeme = get_token();
    STMTLIST();
    if (lexeme != end)
        error();
    lexeme = get_token();
}
```

```
void DECLIST()
{
    if (lexeme != d)
        error();
    lexeme = get_token();
    X();
}
```

Метод рекурсивного спуска

```
void X()
{
    if (lexeme == semi)
    {
        lexeme = get_token();
        DECLIST();
    }
    else if (lexeme != comma)
        error();
}
void STMTLIST()
{
    if (lexeme != s)
        error();
    lexeme = get_token();
    Y();
}
void Y()
{
    if (lexeme == semi)
    {
        lexeme = get_token();
        STMTLIST();
    }
    else if (lexeme != end)
        error();
}
```

Метод рекурсивного спуска

```
int get_token()
{
    char token[132] = "";
    char* tok = token;

    /* Skip spaces */
    while (isspace(*g_prog))
        ++g_prog;

    /*End line marker*/
    if ((char*)NULL == (char*)*g_prog)
        return EOP;

    /* Keywords */
    if (isalpha(*g_prog))
    {
        while (isalpha(*g_prog))
        {
            *tok = *g_prog;
            ++tok;
            ++g_prog;
        }
        *tok = '\\0';
    }
}
```

Метод рекурсивного спуска

```
if (0 == strcmp(token, "begin"))
    return begin;
if (0 == strcmp(token, "comma"))
    return comma;
if (0 == strcmp(token, "end"))
    return end;
if (0 == strcmp(token, "semi"))
    return semi;
if (0 == strcmp(token, "d"))
    return d;
if (0 == strcmp(token, "s"))
    return s;
}
return UNDEF;
}
void error()
{
    printf("Reject.\n");
    exit(EXIT_FAILURE);
}
```

LL(k)-грамматики с $k > 1$

Для построения распознавателей LL(k)-грамматик используются два важных множества, определяемые следующим образом:

$FIRST(k, \alpha)$ – множество терминальных цепочек, выводимых из $\alpha \in (V_T \cup V_N)^*$, укороченных до k символов;

$FOLLOW(k, A)$ – множество укороченных до k символов терминальных цепочек, которые могут следовать непосредственно за $A \in V_N$ в цепочках вывода

Формально эти два множества могут быть определены следующим образом:

$FIRST(k, \alpha) = \{ \omega \in V_T^* \mid \text{либо } |\omega| \leq k \text{ и } \alpha \Rightarrow^* \omega, \text{ либо } |\omega| > k \text{ и } \alpha \Rightarrow^* \omega x, x \in (V_T \cup V_N)^* \},$
 $\alpha \in (V_T \cup V_N)^*, k > 0.$

$FOLLOW(k, A) = \{ \omega \in V_T^* \mid S \Rightarrow^* \alpha A \gamma \text{ и } \omega \in FIRST(k, \gamma), \alpha \in V_T^*, A \in V_N, k > 0.$

Очевидно, что если имеется цепочка терминальных символов $\alpha \in V_T^*$, то $FIRST(k, \alpha)$ – это первые k символов цепочки α .

Доказано, что КСГ G является LL(k)-грамматикой тогда и только тогда, когда выполняется следующее условие:

$\forall A \rightarrow \beta \in P \text{ и } \forall A \rightarrow \gamma \in P (\beta \neq \gamma): FIRST(k, \beta \omega) \cap FIRST(k, \gamma \omega) = \emptyset$ для всех цепочек ω таких, что $S \Rightarrow^* \alpha A \omega$.

Иначе говоря, если существуют две цепочки вывода:

$S \Rightarrow^* \alpha A \gamma \Rightarrow^* \alpha z \gamma \Rightarrow^* \alpha \omega$

$S \Rightarrow^* \alpha A \gamma \Rightarrow^* \alpha t \gamma \Rightarrow^* \alpha \upsilon$

то из условия $FIRST(k, \omega) = FIRST(k, \upsilon)$ следует, что $z = t$.

LL(k)-грамматики с $k > 1$

- **СТУДЕНТАМ:** почему грамматика

$S \rightarrow aS \mid a$

не является LL(1)-грамматикой?

- Почему она является LL(2)-грамматикой?

- Иногда при СА требуется переменное число символов предпросмотра

- Такие грамматики и анализаторы относятся к классу LL(*)

- Подробнее, см. например, - <http://wwwantlr.org/papers/LL-star-PLDI11.pdf>

Дополнительные источники

- LL(1) - [http://ru.wikipedia.org/wiki/LL\(1\)](http://ru.wikipedia.org/wiki/LL(1))
- Ахо, А. Компиляторы: принципы, технологии и инструментарий, 2 издание / А. Ахо, М.Лам, Р. Сети, Дж. Ульман. – М.: Издательский дом «Вильямс», 2008. – 1184 с.
- Метод рекурсивного спуска - http://ru.wikipedia.org/wiki/Метод_рекурсивного_спуска
- LL-анализатор - <http://ru.wikipedia.org/wiki/LL-анализатор>
- The Compiler Generator Cocomac/R - <http://www.ssw.uni-linz.ac.at/Research/Projects/Cocomac/>