Распознаватели для формальных языков

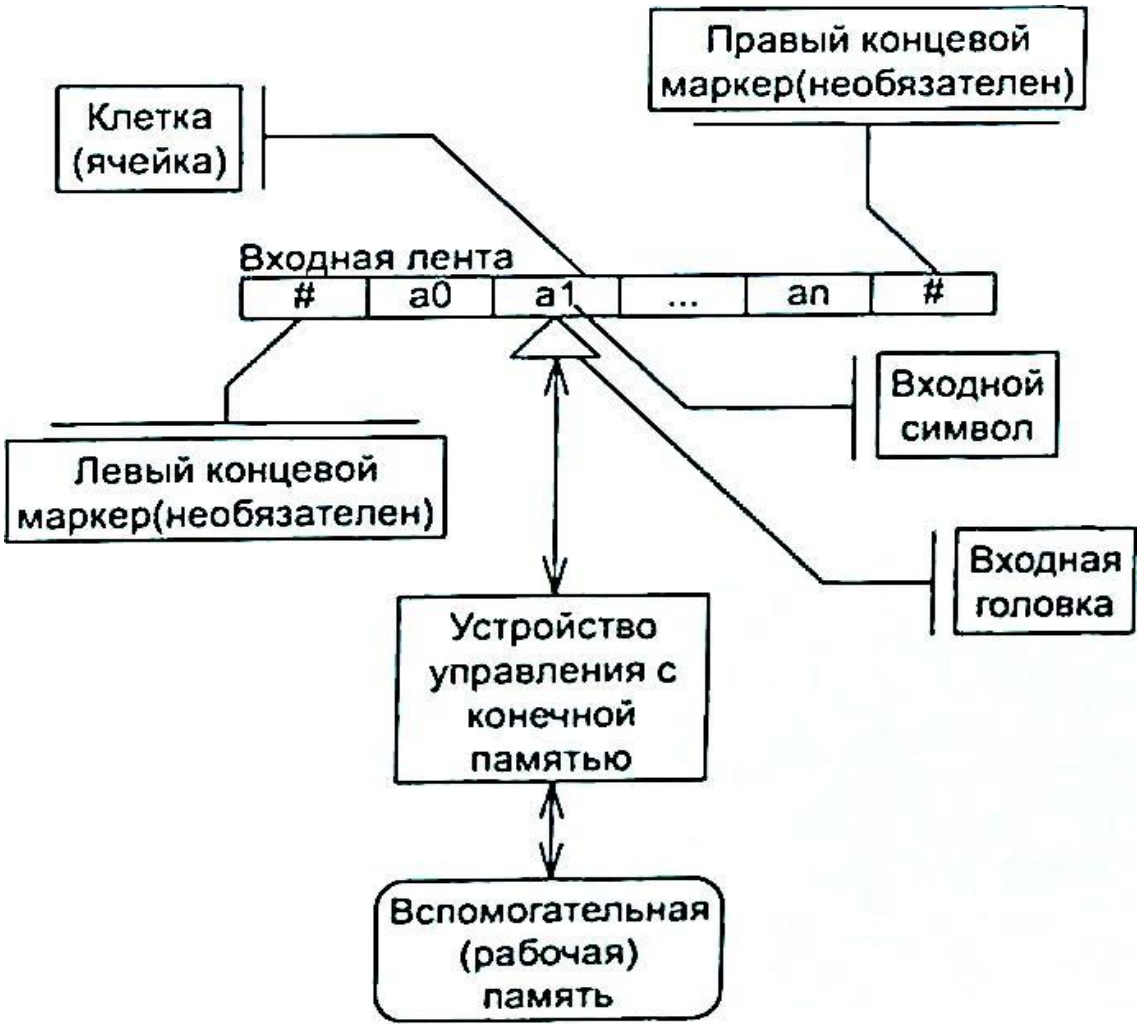
Лекция 2

План лекции

1. Общая схема распознавателя
2. Классификация распознавателей
3. Распознаватели КС языков. Автомат с магазинной памятью (МП-автомат)
4. Синтаксический разбор
5. Методы разбора
6. Нисходящие распознаватели с возвратами
7. Нисходящие распознаватели без возвратов
8. Преобразование КС грамматик

Общая схема распознавателя

Распознаватель – это специальный алгоритм, который позволяет определить принадлежность цепочки символов некоторому языку.



Общая схема распознавателя

В процессе работы распознаватель может выполнять некоторые элементарные операции:

- Чтение очередного символа
- Сдвиг либо входной ленты либо считывающего устройства на заданное количество символов
- Преобразование информации в памяти
- Изменение состояния устройства управления

УУ определяет какая операция будет выполняться на каждом шаге работы распознавателя.

Общая схема распознавателя

Конфигурация распознавателя определяется:

- состоянием устройства управления;
- содержимым цепочки символов и положением считывающей головки в ней;
- содержимым внешней памяти.

Для распознавателя задана начальная конфигурация:

- устройство управления находится в заданном начальном состоянии,
- входная головка читает самый левый символ на входной ленте,
- память либо пуста либо имеет заранее установленное начальное содержимое.

Общая схема распознавателя

Заключительная конфигурация:

- устройство управления находится в одном из состояний, принадлежащем заранее выделенному множеству заключительных состояний,
- входная головка обозревает правый концевой маркер
- иногда требуется, чтобы заключительная конфигурация памяти удовлетворяла некоторым условиям.

Распознаватель допускает входную цепочку α если, начиная с начальной конфигурации, в которой цепочка записана на входной ленте, распознаватель может проделать последовательность шагов, заканчивающихся заключительной конфигурацией.

Классификация распознавателей

- по видам считывающих устройств
  - ✓ односторонние
  - ✓ двусторонние
- по видам УУ
  - ✓ детерминированные
  - ✓ недетерминированные
- по виду внешней памяти
  - ✓ без внешней памяти
  - ✓ с ограниченной внешней памятью
  - ✓ с неограниченной внешней памятью

Сложность распознавателя напрямую зависит от типа языка, входящие цепочки которого могут допускать распознаватели.

Классификация распознавателей

- Распознавателем языка с фразовой структурой является недетерминированный двусторонний автомат с неограниченной памятью (машина Тьюринга).
- Распознавателем КЗ языка является недетерминированный двусторонний автомат с линейно-ограниченной памятью.
- Распознавателем КС языка является недетерминированный односторонний автомат с ограниченной магазинной памятью (МП-автомат).
- Среди всех КС языков выделяют класс КС детерминированных языков.
- Распознавателем регулярного языка является односторонний детерминированный конечный автомат без внешней памяти.

Задача разбора

- На основе имеющейся грамматики некоторого формального языка построить распознаватель этого языка.
- Для КС, регулярных языков известно, что задача разбора разрешима.

Распознаватели КС языков. Автомат с магазинной памятью (МП-автомат)

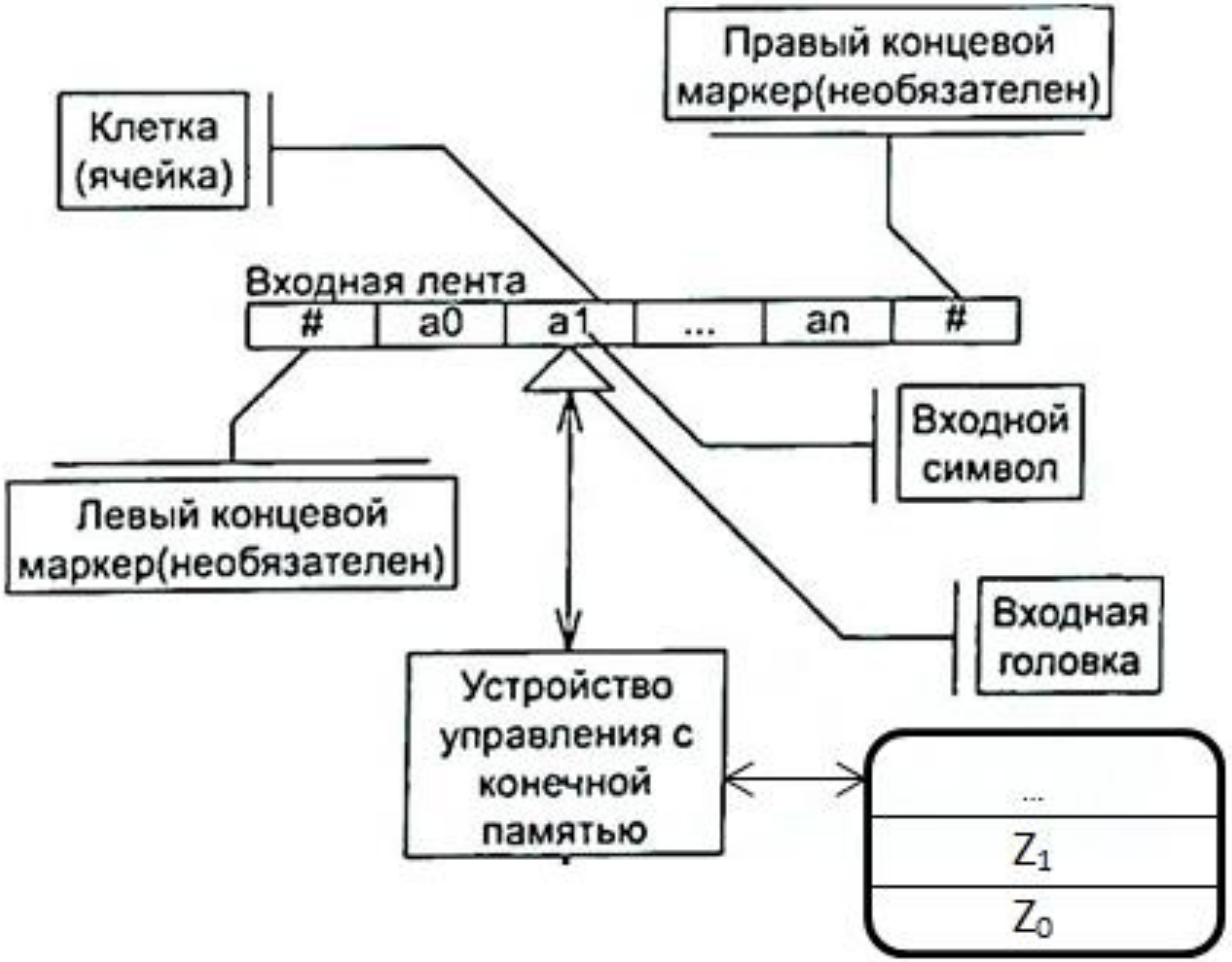
МП –автомат можно представить следующим образом

$R\ (Q,V,Z,\ \delta,\ q_0,\ Z_0,F)$.

- Q – множество состояний
- V – алфавит
- Z – множество магазинных символов $V \leq Z$
- δ - функция переходов, которая отображает

  $Q \times (V \cup \{\varepsilon\}) \times Z$ в подмножество $P(Q \times Z^*)$
- q_0 – начальное состояние
- Z_0 – начальный магазинный символ
- F – непустое множество конечных состояний $F \leq Q$

Распознаватели КС языков. Автомат с магазинной памятью (МП-автомат)



Распознаватели КС языков. Автомат с магазинной памятью (МП-автомат)

- МП-автомат называется недерминированным, если возможен переход из одной конфигурации в более чем одну конфигурацию.

- МП-автомат принимает входную цепочку символов, если он переходит из начальной конфигурации (q_0, Z_0, α), $q_0 \in Q$, $Z_0 \in Z$, в одну из конечных конфигураций (f, z, ε), $f \in F$, $z \in Z$, получив на вход эту цепочку символов.

Распознаватели КС языков. Автомат с магазинной памятью (МП-автомат)

На каждом шаге МП-автомат выполняет операции:

- $\uparrow$ - выталкивает из магазина верхний символ
- $\downarrow A$ – поместить в магазин символ A.
- $\updownarrow XYZ$ – символ X замещается YZ

  Эквивалентно: $\uparrow X \downarrow Y \downarrow Z$
- [t] – УУ переходит в следующее состояние, $t \in Q$
- $\rightarrow$ - входная головка смещается на один символ вправо.

Распознаватели КС языков. Автомат с магазинной памятью (МП-автомат)

Разработать автомат с магазинной памятью для разбора скобочных выражений.

- Множество входных символов: V= { (,) , ⊥}
- Множество магазинных символов: Z = { A, маркер дна ▽)
- Множество состояний: t
- Q = { S }
- q_0 = S
- Z_0 = S
- F = { S }

Распознаватели КС языков. Автомат с магазинной памятью (МП-автомат)

	Конфигурация	Действие
1	(, A, S	↓A, S, →
2	(, ▽, S	↓A, S, →
3	), A, S	↑, S, →
4	), ▽, S	Отвергнуть
5	⊣, A, S	Отвергнуть
6	⊣, ▽, S	Допустить

Распознаватели КС языков. Автомат с магазинной памятью (МП-автомат)

Номер шага	Содержимое стека	Состояние автомата	Остаток входной цепочки	Номер применяемого правила
1	▽	[t]	(( ) ( )) ⊣	2
2	▽ A	[t]	( ) ( )) ⊣	1
3	▽ A A	[t]	) ( )) ⊣	3
4	▽ A	[t]	( )) ⊣	1
5	▽ A A	[t]	)) ⊣	3
6	▽ A	[t]	) ⊣	3
7	▽	[t]	⊣	Допустить

Распознаватели КС языков. Автомат с магазинной памятью (МП-автомат)

Д\З. Разобрать работу МП-автомата

- α = ***()(())***
- β = ***(())***

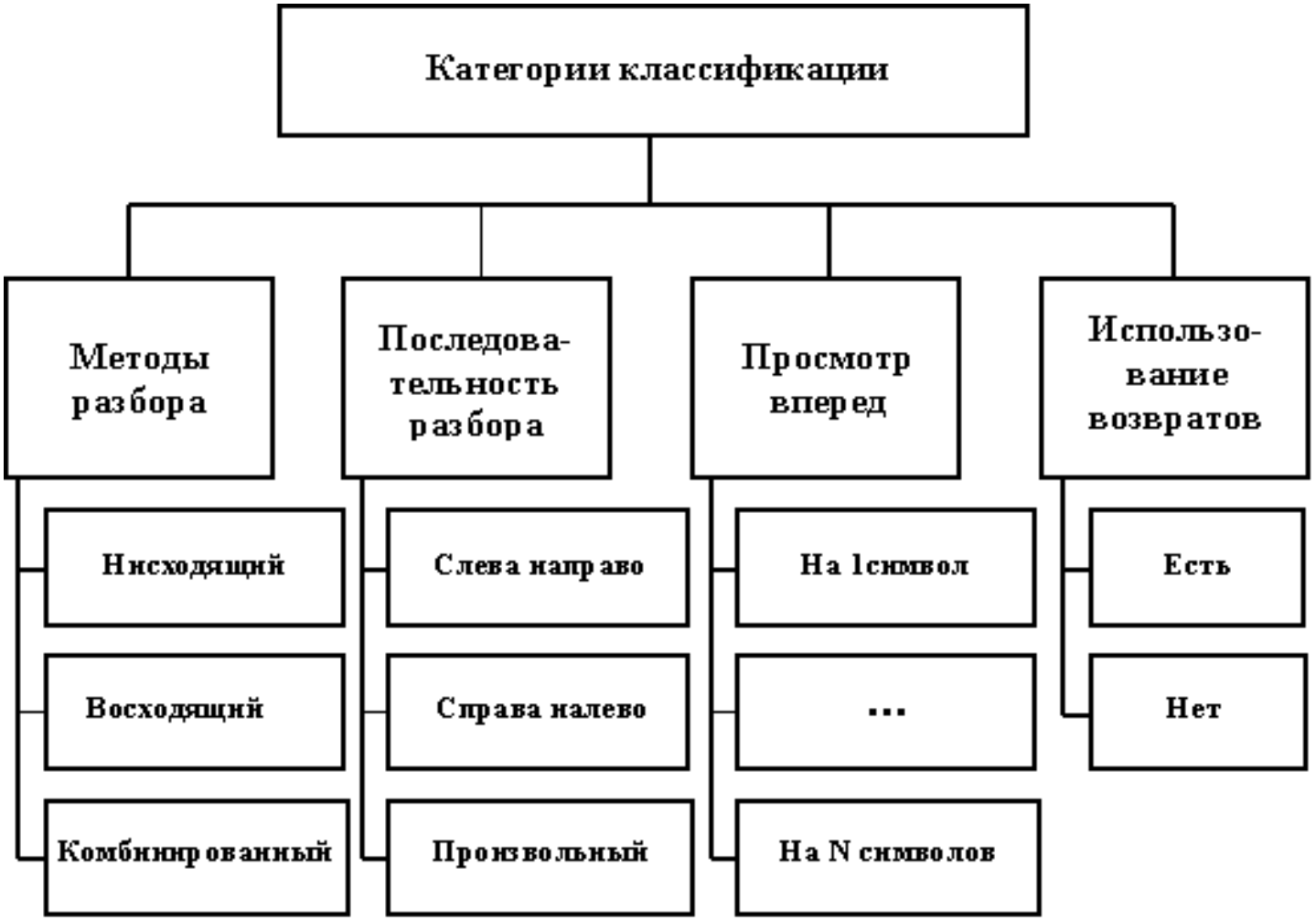
Результат работы представить в виде таблички.

Синтаксический разбор

Если попытаться формализовать задачу на уровне элементарного метаязыка, то она будет ставиться следующим образом:

- Дан язык L(G) с грамматикой G, в которой S - начальный нетерминал.
- Построить дерево разбора входной цепочки $\omega = a_1a_2a_3...a_n$.

Классификация методов организации синтаксического разбора



Методы разбора

- Нисходящий разбор заключается в построении дерева разбора, от корневой вершины. Разбор заключается в заполнении промежутка между начальным нетерминалом (начальный символ грамматики) и символами входной цепочки правилами, выводимыми из начального нетерминала. Подставляемое правило в общем случае выбирается произвольно.

Методы разбора

G = ({S}, {a, +, *}, P, S),

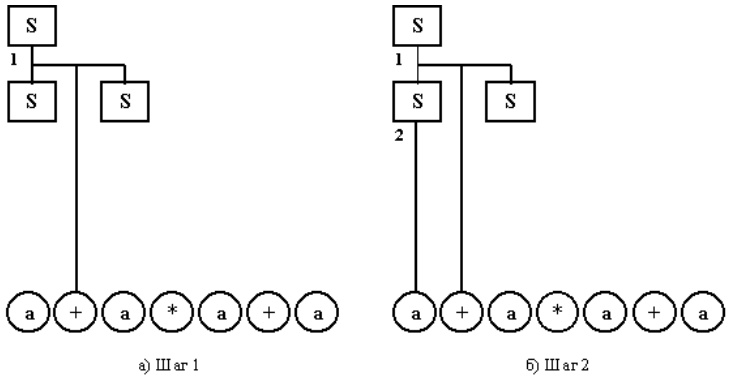
S -> a

S -> S + S

S -> S * S

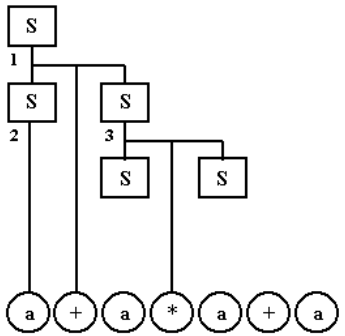
- S => S+S => a+S => a+S*S => a+a*S => a+a*S+S => a+a*a+S => a+a*a+a - левосторонний
- S => S+S => S+a => S*S+a => S*a+a => S+S*a+a => S+a*a+a => a+a*a+a -правосторонний
- S => S*S => S+S*S => S+S*S+S => a+S*S+S => a+a*S+S => a+a*S+a => a+a*a+a - произвольный

Нисходящий разбор слева-направо

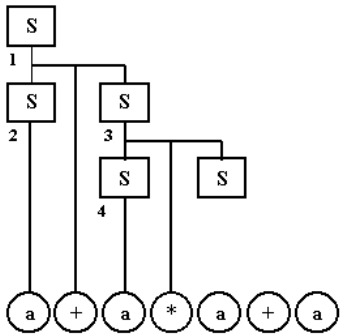


а) Шаг 1

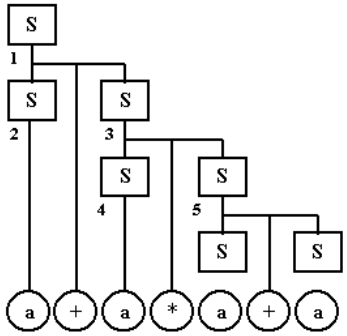
б) Шаг 2



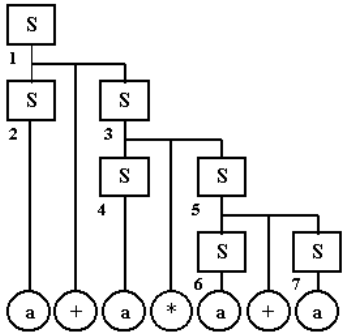
в) Шаг 3



г) Шаг 4

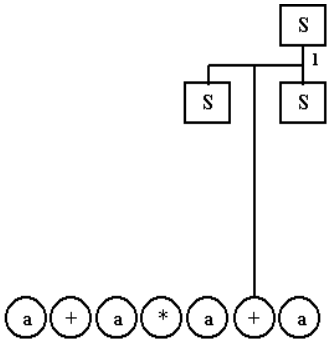


д) Шаг 5

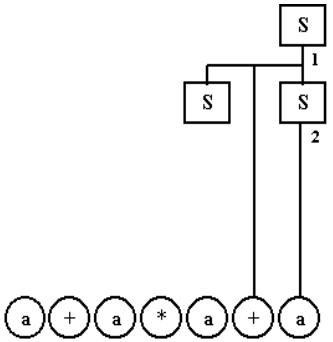


е) Шаги 6-7

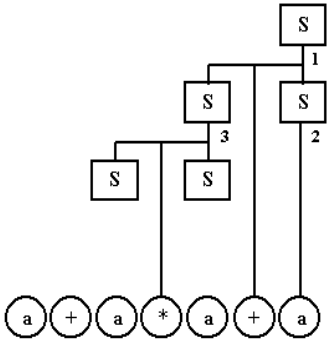
Нисходящий разбор справа-налево



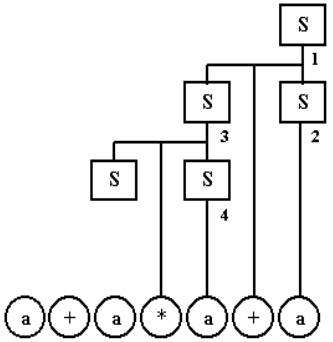
а) Шаг 1



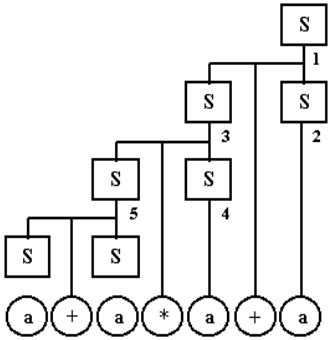
б) Шаг 2



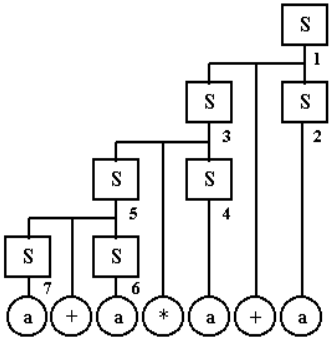
в) Шаг 3



г) Шаг 4

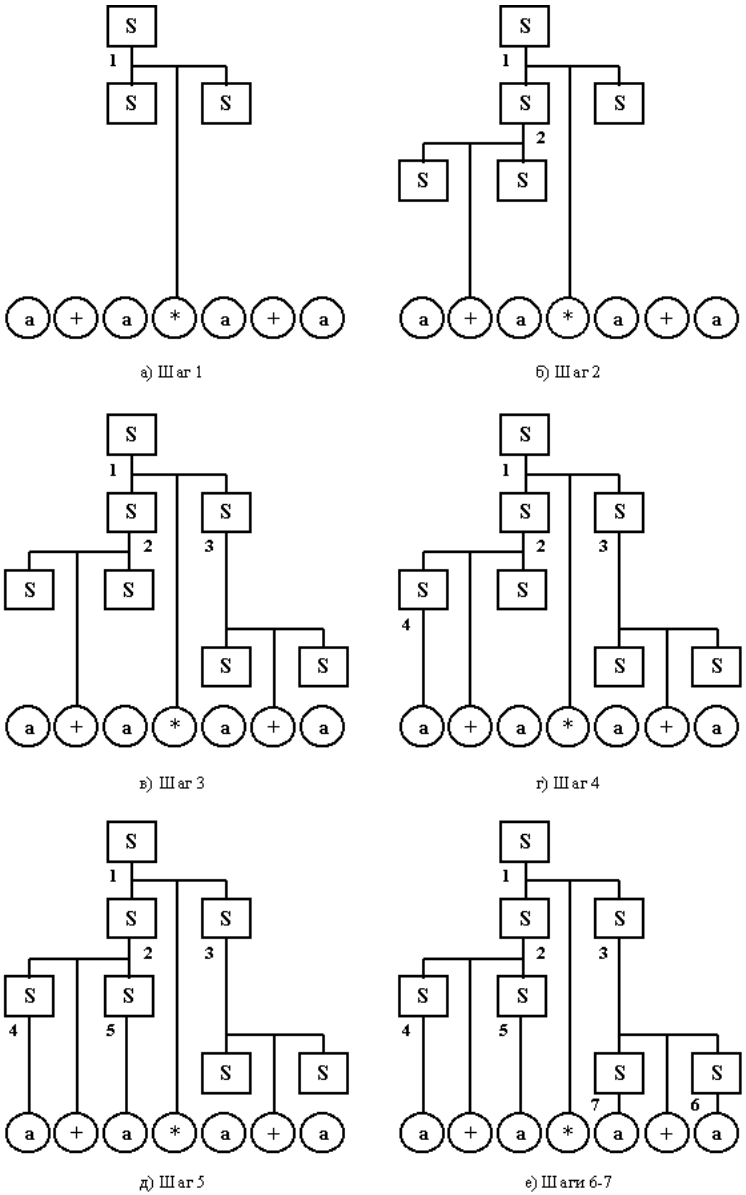


д) Шаг 5



е) Шаги 6-7

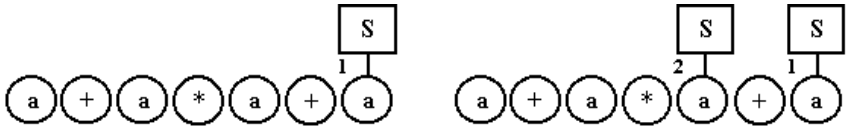
Нисходящий произвольный разбор



Методы разбора

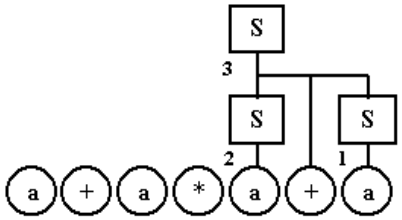
- При восходящем разборе дерево начинает строиться от терминальных листьев путем подстановки правил, применимых к входной цепочке, в общем случае, в произвольном порядке.
- Процесс построения дерева разбора завершается, когда все символы входной цепочки будут являться листьями дерева, корнем которого окажется начальный нетерминал.

Восходящий разбор слева-направо

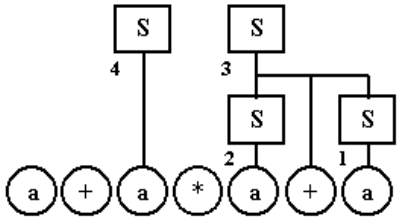


а) Шаг 1

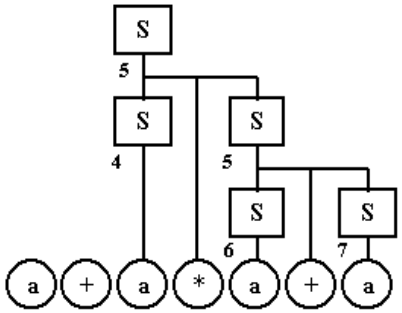
б) Шаг 2



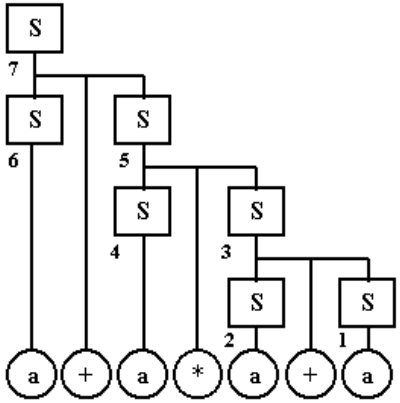
в) Шаг 3



г) Шаг 4

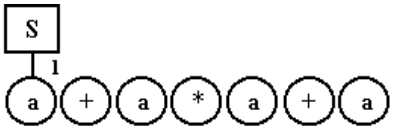


д) Шаг 5

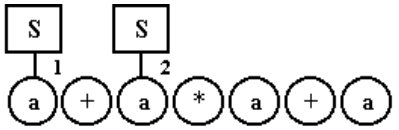


е) Шаги 6-7

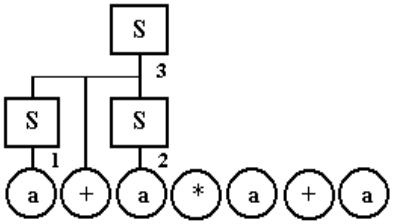
Восходящий разбор справа-налево



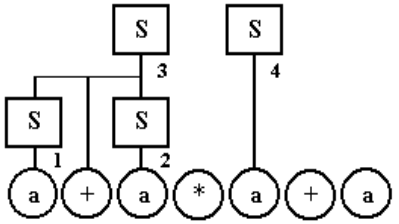
а) Шаг 1



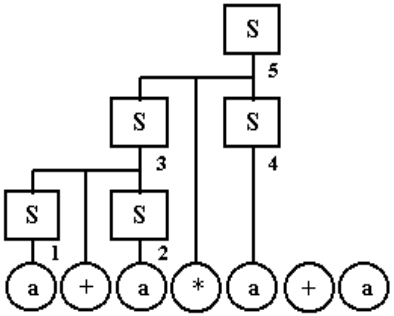
б) Шаг 2



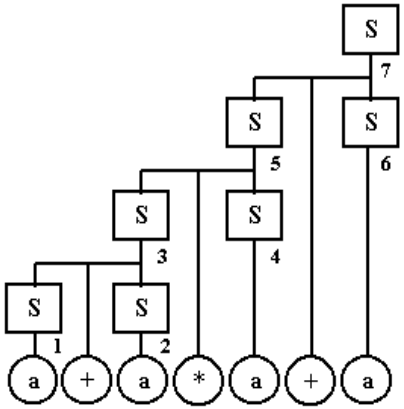
в) Шаг 3



г) Шаг 4

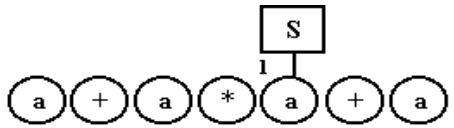


д) Шаг 5

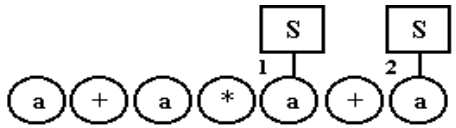


е) Шаги 6-7

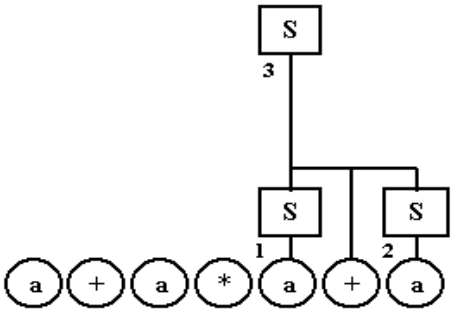
Восходящий произвольный разбор



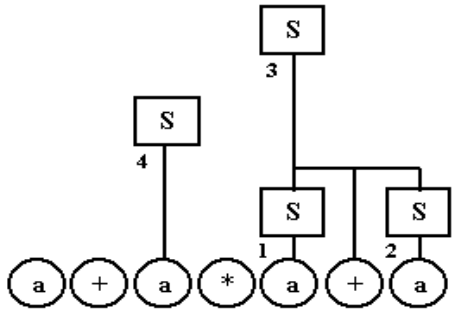
а) Шаг 1



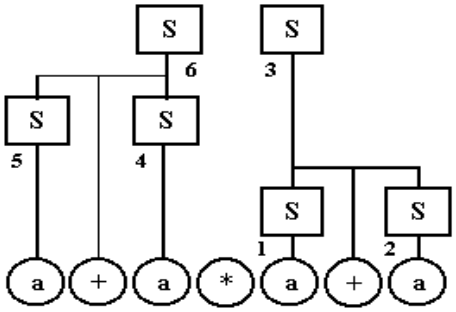
б) Шаг 2



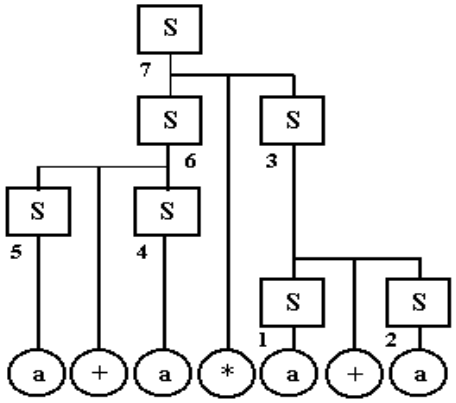
в) Шаг 3



г) Шаг 4



д) Шаги 5-6

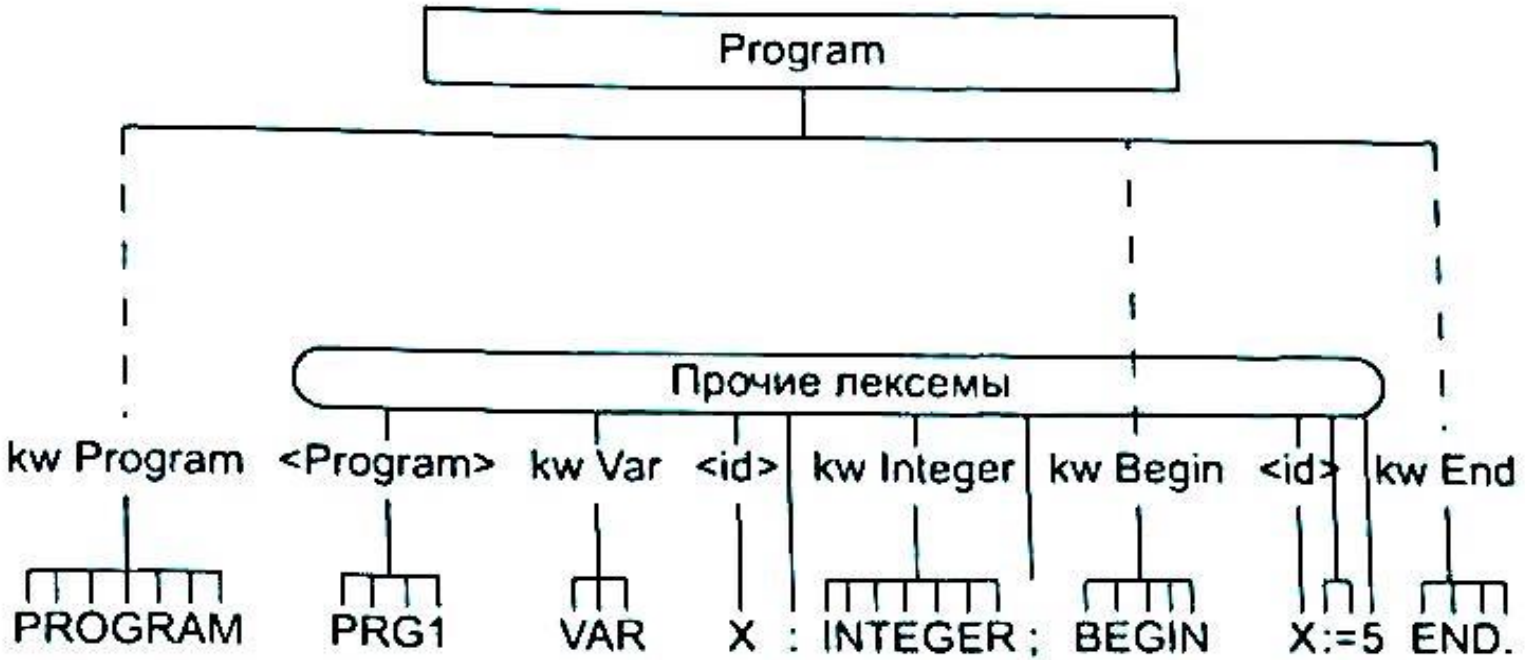


е) Шаг 7

Методы разбора

- Комбинированный разбор может быть реализован тогда, когда процесс распознавания разбивается на два этапа. На одном из них осуществляется нисходящий, а на другом - восходящий разбор.
- Комбинированным можно считать разбор в любом трансляторе, если фазу лексического анализа принять за первый этап, а синтаксического - за второй.

Пример комбинированного разбора



Последовательность разбора

- Повышение эффективности разбора осуществляется разработкой грамматик, специально поддерживающих согласованные между собой метод и последовательность.
- Грамматики предназначенные для нисходящего разбора обычно используются для левостороннего вывода, входная цепочка будет разбираться слева направо (когда порождение новой цепочки на каждом шаге осуществляется для самого левого нетерминала).
- Грамматики, ориентированные на восходящий разбор, обычно оптимизированы под правосторонний вывод, что позволяет, при синтаксическом разборе, осуществлять подстановки нетерминалов справа налево (когда порождение новой цепочки на каждом шаге осуществляется для самого правого нетерминала).

Использование просмотра вперед

- В грамматиках могут встречаться альтернативные правила, начинающиеся с одинаковых цепочек символов. Возникающая неоднородность может быть решена путем предварительного просмотра правила на n символов вперед до той границы, начиная с которой данное правило можно будет отличить от других.
- В КС грамматиках число, определяющее количество символов, анализируемых перед выбором правила подстановки (1,2,...) используется для классификации: КС(1), КС(2).
- На ряду с просмотром вперед используется: преобразование грамматик к однозначным (детерминированным) и анализ с возвратами.

Использование возврата

- Синтаксический разбор с возвратами выполняется аналогично тому, как осуществляется непрямой лексический анализ. Возвраты производятся для альтернативных правил, начинающихся с одинаковых подцепочек.
- Такой подход замедляет разбор.

Использование возврата

Рассмотрим КС-грамматику.

$L = \{a^n b^n \mid n>0\}$

- *ab* *n=1*
- *aabb* *n=2*
- *aaabbb* *n=3*

S -> ASB | ε

A -> a

B -> b

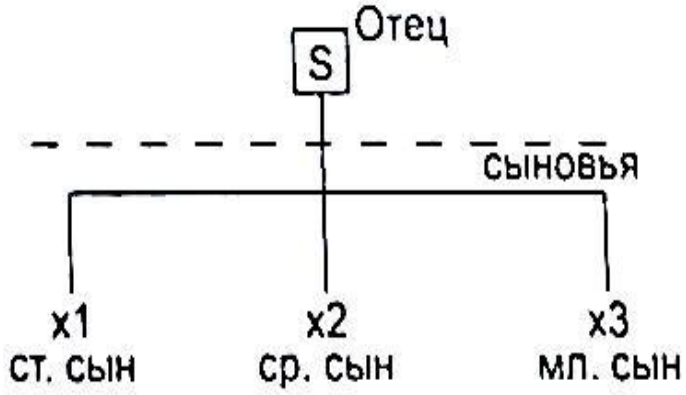
Или

S -> aSb | ab

Нисходящие распознаватели с возвратами

Вообразим, что на любом этапе разбора, в каждом узле уже построенной части дерева находится по одному человеку. Люди, которые находятся в терминальных узлах, занимают места соответственно символам предложения.

Участок дерева разбора



Некоему человеку надлежит провести разбор предложения ω.

- Ему необходимо отыскать вывод $S \Rightarrow^{+} \omega$, где S – начальный символ.

- Пусть для S существуют правила

$$S ::= X_1 X_2 .. X_n \mid Y_1 Y_2 .. Y_m \mid Z_1 Z_2 .. Z_k$$

- Сначала человек пытается определить правило $S ::= X_1 X_2 .. X_n$. Если нельзя построить дерево, используя это правило, он делает попытку применить второе правило $S ::= Y_1 Y_2 .. Y_m$. В случае неудачи он переходит к следующему правилу и т.д.

Как ему определить, правильно он выбрал непосредственный вывод $S ::= X_1 X_2 .. X_n$?

- Если вывод правилен, то для некоторых цепочек x_i будет иметь место $\omega = x_1 x_2 .. x_n$, где $X_i => + x_i$, для $i=1,...,n$.
- Прежде всего, человек, выполняющий разбор, возьмет себе приемного сына M_1, который должен будет найти вывод $X_1 =>^* x_1$, такого, что $\omega = x_1...$
- Если сыну M_1 удастся найти такой вывод, он (и любой из сыновей, внуков и т.д.) закрывает цепочку x_1в предложении ω и сообщает своему отцу об успехе.

- Тогда его отец усыновит M_2, чтобы тот нашел вывод $X_2 \Rightarrow {}^*x_2$, где $\omega = x_1x_2...$ и ждет ответа от него и т.д.
- Как только сообщил об успехе сын M_{i-1},он усыновит еще и M_i, чтобы тот нашел вывод $X_i \Rightarrow {}^*x_i$.
- Сообщение об успехе, пришедшее от сына M_n, означает что разбор предложения закончен.

Как же действует каждый из M_i?

- Положим, целью M_i является терминал t, такой, что $\omega = x_1 x_2 .. x_{i-1} t..$,где символы в $x_1, x_2, ..., x_{i-1}$ уже закрыты другими людьми. M_i проверяет, совпадает ли очередной незакрытый символ t с его целью X_i. Если это так, он закрывает этот символ и сообщает об успехе. Если нет, сообщает об неудаче.
- Если цель M_i – нетерминал X_i, то M_i поступает точно так же, как и его отец. Он начинает проверять правые части правил, относящихся к нетерминалу, и, если необходимо, тоже усыновляет или отрекается от сыновей. Если все его сыновья сообщают об успехе то M_i в свою очередь сообщает об успехе отцу.

- Если отец просит M_i найти другой вывод, а целью является терминальный символ, то M_i сообщает о неудаче, так как другого такого вывода не существует. В противном случае M_i просит своего младшего сына найти другой вывод и реагирует на его ответ также, как и раньше. Если все сыновья сообщат о неудаче, он сообщит о неудаче своему отцу.
- Каждый человек должен помнить о своей цели, о своем отце, сыновьях и свое место во входной цепочке и грамматике.

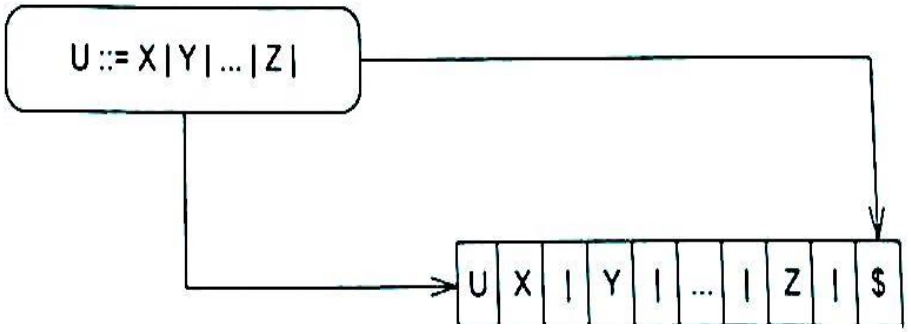
Реализация нисходящего распознавателя с возвратами

Форма, которая будет использоваться для записи правил

G = ({i, +, *, (,)}, {S, E, T, F}, P, S)

P:
1) S ::= E#
2) E ::= T + E
3) E ::= T
4) T ::= F * T
5) T ::= F
6) F ::= (E)
7) F ::= i

| - альтернатива
| $ - признак конца правила
| $ % - признак конца грамматики



GRAMMAR

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
⊔	S	E	#	|	$	E	T	+	E	|	T	|	$	T	F	*	T	|	F	|

S ::= E #

E ::= T + E | T

21	22	23	24	25	26	27	28	29
$	F	(	E	)	|	i	|	$

Принятые обозначения

- char input []; - строка содержит входную цепочку символов;
- char grammar []; - массив с грамматикой;
- int j; - индекс самого левого незакрытого терминала входной цепочки input[j];
- int i; - индекс в массиве grammar определяющий цель с которой работает человек в данный момент;
- struct node {

  char goal; - цель

  int fat; - "имя" отца

  int son; - "имя" младшего из сыновей

  int bro; - "имя" его брата

  int i; - индекс в массиве grammar определяющий цель, с которой работает человек в данный момент }
- int v; - количество элементов в стеке;
- int c; - "имя" человека (индекс в стеке);
- #define MAX_LEVEL 50
- node S[MAX_LEVEL]; - стек;

Принятые обозначения

- Понятия относящиеся к человеку, работающему в данный момент (находится на уровне c)

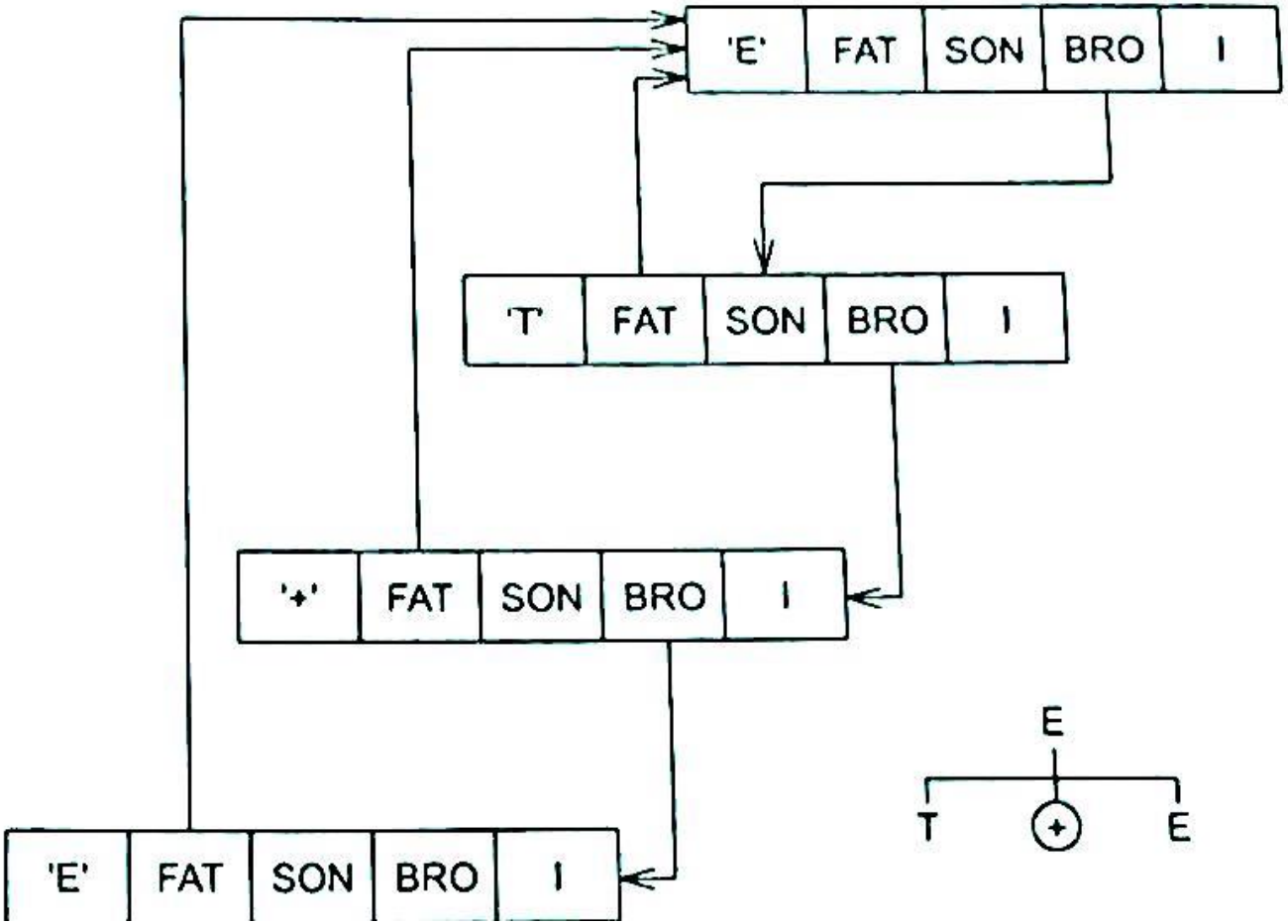
```
#define GOAL    S[ c ] . goal
#define FAT     S[ c ] . fat
#define SON     S[ c ] . son
#define BRO     S[ c ] . bro
#define I       S[ c ] . i
```

S(v) = (GOAL, FAT, SON, BRO, I) - в стек на уровень v заносится информация о текущей вершине разбора.

Функции:

- int terminal (char g) - определяет, является ли g терминалом, если да, то возвращает 1 иначе 0.
- int index (char g) - возвращает индекс правой части для цели g в массиве grammar.
- int stop(int result) - прекращает разбор и возвращает результат разбора, т.е. является ли данная цепочка предложением или нет.

Структура "семьи"



Алгоритм в псевдокоде

```
Начальная установка:
S(l) = ('S', 0, 0, 0, 0); c = 1; v = 1; j = 1;
        goto новый человек ;

новый человек:
if (terminal(GOAL))
        if (input[j] == GOAL)
        {
                j++; goto успех
        }
        else goto неудача
I = index(GOAL);     // индекс правой части для GOAL

цикл:
if (grammar[I] == '|')
        if (FAT != 0) goto успех ;
        else stop('сообщение');        //  предложение языка
if (grammar[I] == '$')                 // конец правила
        if (FAT != 0) goto неудача
        else stop('сообщение);         // не предложение языка
```

Алгоритм в псевдокоде

```
// очередная установка
v++;
S(v) = (grammar[I], 0, c, 0, SON);
SON = v; c = v; goto  новый человек

успех:
c = FAT; I++; goto цикл

неудача:
c = FAT; v--; son = S(son).bro; goto еще раз

еще раз:
if (SON == 0) {while (grammar [I++] != '|');          // переход к следующему правилу
goto цикл                                  // просьба к сыну повторить попытку выбора
 }

I--; c = SON;
if (!terminal(GOAL)) goto еще раз;        // цель терминал, вывод построить нельзя
j--; goto неудача ;
```

INPUT

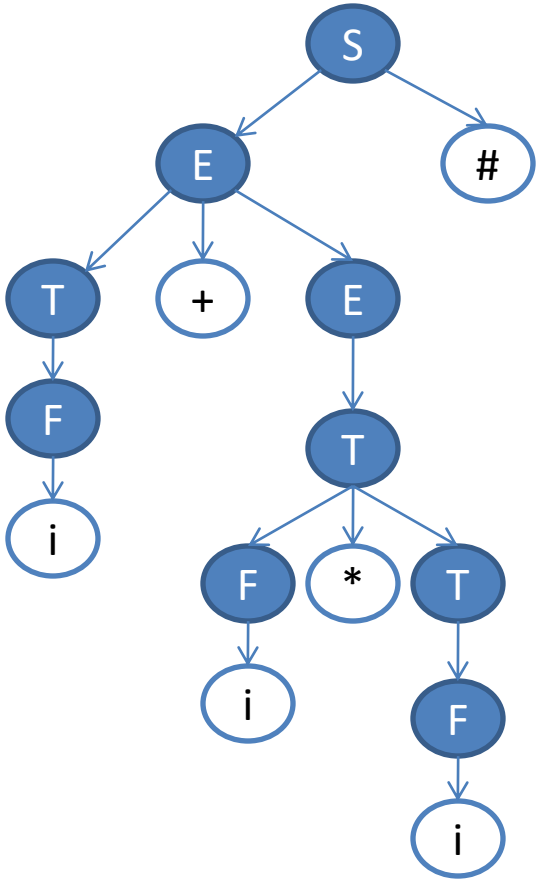
0	1	2	3	4	5	6	
⊔	i	+	i	*	i	#	

input string

Пример

GRAMMAR

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	
⊔	S	E	#	|	$	E	T	+	E	|	T	|	$	T	F	*	T	|	F	|	$	F	(	E	)	|	i	|	$	



N	Стек	GOAL	I	FAT	SON	BRO	c, V, j	вх. симв.
1	1	S	0	0	0	0	c= 1; v = 1; j = 1	i
2	1	S	2	0	0	0	v = 2	
3	2	E	0	1	0	0		
4	1	S	2	0	2	0	c = 2	
5	2	E	7	1	0	0	v = 3	
6	3	T	0	2	0	0		
7	2	E	7	1	3	0	c = 3	
8	3	T	15	2	0	0	v = 4	
9	4	F	0	3	0	0		
10	3	T	15	2	4	0	c = 4	
11	4	F	23	3	0	0	v = 5	
12	5	(	0	4	0	0		
13	4	F	23	3	5	0	c = 5	
14	5	(	0	4	0	0	c = 4, v = 4	
15	4	F	23	3	0	0		
16	4	F	27	3	0	0	v = 5	
17	5	i	0	4	0	0		
18	4	F	27	3	5	0	c = 5, j = 2	
19	5	i	0	4	0	0	j = 2, c = 4	+
20	4	F	28	3	5	0	c=3	

Нисходящие распознаватели без возвратов

- Алгоритм работы МП-автомата не требует возврата на предыдущий шаг и обладает линейными характеристиками от длины входной цепочки.
- В случае не успеха выполнения алгоритма входная цепочка однозначно не принимается и повторная итерация разбора не принимается.
- Выбор одного из возможных альтернатив является выбор ее на основе символа a є VT, обозреваемого считывающей головкой автомата на каждом шаге его работы.

Левосторонний разбор по методу рекурсивного спуска

- Для каждого A є VN, строится своя процедура разбора, которая получает на вход цепочку символов α и положение считывающей головки.
- Если для A определено более одного правила, то процедура разбора ищет среди множества правил вида

  A -> aγ, a є VT, γ є $(VT \cup VN)^*$ правила, первый символ которого совпадал бы с текущим входным символом

  a = α[i]:

  - ✓ Если такого правила нет, то алгоритм прекращается и цепочка не является цепочкой языка,
  - ✓ Если правило найдено и единственное, то запоминается номер правила, считывающая головка перемещается вправо (i++), а для каждого нетерминала цепочки γ вызывается соответствующая процедура разбора.

Условия применимости РС-метода

- либо A -> α, где $\alpha \in (VT \cup VN)^*$ и это единственное правило вывода для этого нетерминала;
- либо A > $a_1\alpha_1 \mid a_2\alpha_2 \mid \ldots \mid a_n\alpha_n$, где $a_i \in VT$ для всех i = 1,2,...,n; $a_i \neq a_j$ для $i \neq j$; $\alpha_i \in (VT \cup VN)^*$, т. е. если для нетерминала A правил вывода несколько, то они должны начинаться с терминалов, причем все эти терминалы должны быть различными.

Этим условиям удовлетворяют незначительное количество КС-грамматик, это достаточные, но необязательные условия.

Пример реализации РС-метода

G ({a,b,c}, {A,B,C,S}, P, S)

P:
1) S -> aA
2) S -> bB
3) A -> a
4) A -> bA
5) A -> cC
6) B -> b
7) B -> aB
8) B -> cC
9) C -> AaBb

```
int main (int argc, char* argv[]) {
        fin = fopen(argv[1], “2”);
        gc();
        if ( S() )  printf (“Success\n”);
        else  printf (“Error\n”);
        fclose(fin);
        return 1;
}

  // 1) S -> aA
  // 2) S -> bB
  int S () {
        int rc=0;
        if (c==’a’) {
                R.enque(1); gc();
                rc=A();
        } else if (c==’b’) {
                R.enque(2); gc();
                rc=B();
        }
  return (rc);
  }
```

```
extern char c;
extern file *fin;
char gc();
queue R;



// 3) A -> a
// 4) A -> bA
// 5) A -> cC
int A () {
        int rc=0;
        if (c==’a’) {
                R.enque(3); gc();
                rc=1;
        } else if (c==’b’) {
                R.enque(4); gc();
                rc=A();
        } else if (c==’c’) {
                R.enque(5); gc();
                rc=C();
        }
return (rc);
}
```

```
// 6) B -> b
// 7) B -> aB
// 8) B -> cC

int B () {
        int rc=0;
        if (c==’b’) {
                R.enque(6); gc();
                rc=1;
        } else if (c==’a’) {
                R.enque(7); gc();
                rc=B();
        } else if (c==’c’) {
                R.enque(8); gc();
                rc=C();
        }
return (rc);
}
```

```
// 9) C -> AaBb
int C () {
        int rc=0;
        R.enque(9);
        rc=A();
        if (rc) {
                if (c==’a’) {
                        gc();
                        rc=B();
                        if (rc) {
                                if (c==’b’) {
                                gc();
                                rc=1;
                                }
                        }
                else rc=0;
                }
        }
return (rc);
}
```

Преобразование КС грамматик

- Для КС-грамматик невозможно проверить их однозначность и эквивалентность. Правила КС-грамматик преобразовывают к заранее заданному виду, чтобы получить эквивалентную грамматику.
- Все преобразования можно разбить на две группы:
  - ✓ преобразования, связанные с исключением из грамматики тех правил и нетерминалов, без которых она может существовать (ведет к упрощению правил);
  - ✓ преобразования, в результате которых изменяется вид и состав правил грамматики (не связано с упрощениями).

Приведенные грамматики

- Приведенные КС-грамматики – это КС-грамматики, которые не содержат недостижимых и бесполезных символов, циклов, ε-правил.
- Для того, чтобы преобразовать произвольную КС-грамматику к приведенному виду необходимо:
  - ✓ удалить все бесполезные символы;
  - ✓ удалить все недостижимые символы;
  - ✓ удалить ε-правила;
  - ✓ удалить цепные правила или циклы.

Удаление бесполезных символов

- Символ $A \in VN$ называется бесполезным в грамматике $G = (VT, VN, P, S)$, когда из него нельзя вывести ни одной терминальной цепочки, т.е. если множество $\{ \alpha \in VT^* \mid A \Rightarrow \alpha \}$ пусто.

- **Д/З** Алгоритм удаления бесполезных символов **(мет. Руденко)**

Алгоритм удаления бесполезных символов

Вход: КС-грамматика G = (VT, VN, P, S).

Выход: КС-грамматика G' = (VT, VN', P', S), не содержащая бесплодных символов, для которой L(G) = L(G’).

Метод:

- Рекурсивно строим множества N_0, N_1, ...
- $N_0 = \varnothing$, i = 1.
- $N_i = \{A \mid (A \to \alpha) \in P \text{ и } \alpha \in (N_{i-1} \cup VT)^*\} \cup N_{i-1}$.
- Если $N_i \neq N_{i-1}$, то i = i + 1 и переходим к шагу 2, иначе $VN' = N_i$; P' состоит из правил множества P, содержащих только символы из $VN' \cup VT$;

Алгоритм удаления бесполезных символов

G=({a,b}, {S,A,B,C},S,P)

P: S -> aA | bB

A -> bAa

B -> aB | bS | a | b

C -> BaA

1. $N_0 = \varnothing$, i = 1
2. $N_1 = \{B\}$, $N_1 \neq N_0$, i = 2
3. $N_2 = \{B,S\}$, $N_2 \neq N_1$, i = 3
4. $N_3 = \{B,S\}$, $N_3 = N_2$

A и C – бесполезные символы, все правила, содержащие вхождения этих символов удаляются:

S -> bB

B -> aB |bS| a |b

Удаление недостижимых символов

- Символ $x \in (VT \cup VN)$ называется недостижимым в грамматике G = (VT, VN, P, S), если он не появляется ни в одной сентенциальной форме этой грамматики.

  $x \in \alpha$, где $\{ \alpha \mid S \Rightarrow \alpha, \alpha \in (VT \cup VN) \} \neq 0$

  Пример. G ({a, b}, {S, A, B}, P, S);

  P: S -> a | aA

  A -> b | bA

  B -> b

- **Д/З** Алгоритм удаления недостижимых символов (**мет. Руденко**)

Алгоритм удаления недостижимых символов

- Вход: КС-грамматика G = (VT, VN, P, S)
- Выход: КС-грамматика G' = (VT', VN', P', S), не содержащая недостижимых символов, для которой L(G) = L(G').
- Метод:
  1. $V_0 = \{S\}$; i = 1.
  2. $V_i = \{x \mid x \in (VT \cup VN)$, в P есть $A \to \alpha x \beta$ и $A \in V_{i-1}$, $\alpha, \beta \in (VT \cup VN)^*\} \cup V_{i-1}$.
  3. Если $V_i \neq V_{i-1}$, то i = i + 1 и переходим к шагу 2, иначе $VN' = V_i \cap VN$; $VT' = V_i \cap VT$; P' состоит из правил множества P, содержащих только символы из Vi.

Пример

G = ({a,b,c,d}, {A, B, C, D, E, F, G, S}, P, S)

P:
S -> aAB | E
A -> aA | bB
B -> ACb | b
C -> A | bA | cC | aE
E -> cE | aE | Eb | ED | FG
D -> a | c | Fb
F -> BC | EC | AC | Fd
G -> Ga | Gb

Пример работы алгоритма

1. $N_0 = \varnothing$, i=1
2. N_1 = {B, D}, i=2, $V_0 \neq V_1$
3. N_2 = {B, D, A}, i=3, $V_1 \neq V_2$
4. N_3 = {B, D, A, S, C}, i=4, $V_2 \neq V_3$
5. N_4 = {B, D, A, S, C, F}, i=5, $V_3 \neq V_4$
6. N_5 = {B, D, A, S, C, F}, i=5, $V_4 = V_5$
7. VN' = V_5 = { B, D, A, S, C, F },

VT' = VT

P’:
S -> aAB
A -> aA | bB
B -> ACb | b
C -> A | bA | cC
D -> a | c | Fb
F -> BC | AC | Fd

1. V_0 = {S}, i=1
2. V_1 = {S, a, A, B}, i=2, $V_0 \neq V_1$
3. V_2 = {S, a, A, B, b, C}, i=3, $V_1 \neq V_2$
4. V_3 = {S, a, A, B, b, C, c}, i=3, $V_2 \neq V_3$
5. V_4 = {S, a, A, b, B, C, c}, i=4, $V_3 = V_4$
6. VN'' = V_5 = { B, A, S, C}

VT'' = {a, b, c}

P'':
S -> aAB
A -> aA | bB
B -> ACb | b
C -> A | bA | cC

Устранение ε-правил

- Грамматика G называется грамматикой без ε-правил, если в ней не существует правил вида A -> ε, A ≠ S, и может присутствовать только одно правило S -> ε, в том случае, если пустая цепочка принадлежит языку ε ∈ L (G), и при этом нетерминал S не встречается в правой части ни одного правила грамматики.

Устранение ε-правил

Алгоритм.

1. $V_0 = \{A \mid (A \to \varepsilon) \in P\}$; i = 1.

2. $V_i = V_{i-1} \cup \{A \mid (A \to \alpha) \in P, \alpha \in V_{i-1}\}$.

3. Если $V_i \neq V_{i-1}$, то i=i+1, переход к шагу 2, иначе к шагу 4.

4. VN' = VN, VT’ = VT, в P' входят все правила из P кроме правила A -> ε.

5. Если $(A \to \alpha) \in P$ и $\alpha \in V_i^*$, то на основании цепочки α строим множество цепочек α' путем исключения из α всех возможных комбинаций символов из V_i.

6. Если $S \in V_i$, тогда добавляем S' в множество VN' и в P’: S' -> ε | S, иначе S' = S.

Пример

G ({a, b, c}, {A, B, C, S}, P, S)

P: S -> AaB | aB | cC

A -> AB | a | b | B

B -> Ba | ε

C -> AB | c

1. V_0 ={ B }, i = 1
2. V_1 = { B, A }, i=2, $V_0 \neq V_1$
3. V_2 = { B, A, C }, i=3, $V_1 \neq V_2$
4. V_3 = { B, A, C }, i=4, $V_2 \neq V_3$

VN' = { A, B, C, S } VT' = {a, b, c}

P': S -> AaB | Aa | aB | cC | a | c

A -> AB | a | b | B

B -> Ba | a

C -> AB | A | B | c

Устранение цепных правил

- Циклом или циклическим выводом грамматики G называется вывод $A =>^* A$, $A \in VN$.
- Циклы возможны в том случае, если в КС грамматике присутствует цепное правило $A \to B$, $A, B \in VN$.

Алгоритм.

Для каждого нетерминального символа x строится специальное множество цепных символов N^x. Для каждого нетерминала из множества VN повторяются шаги 1-4, затем переходим к шагу 5.

1. $N_0^x = \{x\}$, i=1
2. $N_i^x = N_{i-1}^x \cup \{ B \mid (A \to B) \in P, A \in N_{i-1}^x \}$, i=1
3. Если $N_i^x \neq N_{i-1}^x$, то i=i+1, переход к шагу 3, иначе $N^x = N_i^x - \{x\}$, переход к шагу 1.
4. VN' = VN, VT' = VT, в P' входят все правила из P кроме правила $A \to B$.
5. Для всех правил $(A \to \alpha) \in P$, если $A \in N^B$, $A \neq B$ в P’ добавляем $B \to \alpha$.

Пример

G ({a, b, c}, {A, B, C, S}, P, S)

P: S - > AaB | Aa | aB | cC | a | c

A -> AB | a | b | B

B -> Ba | a

C -> AB | A | B | c

1. $N_0^S = \{S\}$, i=1
2. $N_1^S = \{S\}$, $N_1^S = N_0^S$, $N_1^S = \varnothing$
3. $N_0^A = \{A\}$, i=1
4. $N_1^A = \{A, B\}$, $N_1^A \neq N_0^A$, i = 2
5. $N_2^A = \{A, B\}$, $N_2^A = N_1^A$, $N_2^A = \{ B \}$
6. $N_0^B = \{B\}$, i=1
7. $N_1^B = \{B\}$, $N_1^B = N_0^B$, $N_1^B = \varnothing$
8. $N_0^C = \{C\}$, i=1
9. $N_1^C = \{C, A\}$, $N_1^C \neq N_0^C$, i = 2
10. $N_2^C = \{C, A, B\}$, $N_2^C \neq N_1^C$, i = 3
11. $N_3^C = \{C, A, B\}$, $N_2^C = N_1^C$, $N_3^C = \{ A, B \}$

VN' = { A, B, C, S } VT' = {a, b, c}

P': S -> AaB | Aa | aB | cC | a | c

A -> AB | a | b | Ba

B -> Ba | a

C -> AB | Ba | c | a | b

Устранение левой рекурсии

- Нетерминальный символ A грамматики G называется рекурсивным, если для него существует вывод $A \Rightarrow^{+} \alpha A \beta$, $\alpha, \beta \in (VT \cup VN)^*$:
  - A - леворекурсивный, если $\alpha = \varepsilon$, $\beta \neq \varepsilon$
  - A - праворекурсивный, если $\alpha \neq \varepsilon$, $\beta = \varepsilon$.
- КС грамматика может быть как лево- так праворекурсивной, а также может быть левоправорекурсивной относительно разных нетерминалов.

Устранение левой рекурсии

1. $N = \{ A_1, A_2 \dots A_n\}$ i=1, n – количество нетерминалов

2. Рассмотрим все правила для A_i. Если эти правила не содержат левой рекурсии, то переносим их в P', символ A_i добавляем в множество VN'.

Иначе если $A_i \to A_i \alpha_1 \mid A_i \alpha_2 \mid \dots \mid A_i \alpha_m \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_p$,

где ни одна цепочка β_j не начинается с символа A_k $1 \le j \le p$, $k \le i$.

Вместо этого правила во множество P' дописывается правило вида

$A_i \to \beta_1 \mid \beta_2 \mid \dots \mid \beta_p \mid \beta_1 A_i' \mid \beta_2 A_i' \mid \dots \mid \beta_p A_i'$

$A_i' \to \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_m \mid \alpha_1 A_i' \mid \alpha_2 A_i' \mid \dots \mid \alpha_m A_i'$

Если i=n, то грамматика G' построена, иначе i=i+1; j=1, переходим к шагу 4.

Устранение левой рекурсии

3. Для устранения косвенной левой рекурсии.

4. Для символа A_j во множестве правил P' заменить все правила вида:

$A_i \rightarrow A_j \alpha$, где $\alpha \in (VT \cup VN)^*$

$A_i \rightarrow \beta_1 \alpha \mid \beta_2 \alpha \mid \ldots \mid \beta_m \alpha$ причем

$A_j \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$ все правила для A_j

Т.к. правая часть нетерминала A_j не может начинаться с нетерминального символа A_i, то и правая часть правил для нетерминала A_i будет начинаться с этого символа.

5. Если j=i-1, то переход к шагу 2, иначе j=j+1, переход к шагу 4.

6. $S' = A_n$

Пример

G = ({a, b, +, *, (,)}, {F, T, E}, P, E)

P: E -> E+T | T

T -> T * F | F

F -> (E) | a | b

$A_1\ A_2\ A_3$ n=3

A_1 -> A_1+ A_2 | A_2 i=1 n=3

A_2 -> A_2 * A_3 | A_3

A_3 -> (A_1) | a | b

P': E -> T | E'

E' -> +T | +T E'

T -> F | T'

T' -> *F | *F T'

F -> (E) | a | b

P': A_1 -> $A_1\alpha_1$ | β_1

A_1 -> A_2 | $A_2\ A_1$'

A_1' -> +A_2 | +$A_2\ A_1$'

A_2 -> $A_2\alpha_2$ | β_2

A_2 -> A_3 | $A_3\ A_2$'

A_2' -> *A_3 | *$A_3\ A_2$'

S' = A_1'

Устранение левой факторизации

Если в грамматике существуют правила вида

$A \to a\alpha_1 \mid a\alpha_2 \mid \ldots \mid a\alpha_n \mid \beta_1 \mid \ldots \mid \beta_m$, где $a \in VT$, α_i, $\beta_j \in (VT \cup VN)^*$

и входная строка начинается с непустой строки, выводимой из a, то неизвестно разворачивать по $a\alpha_1$ или $a\alpha_2$. Можно преобразовать правила вывода данного нетерминала объединив правила вывода с общими началами в одно правило:

$A \to aA' \mid \beta_1 \mid \ldots \mid \beta_m$

$A' \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$

Пример

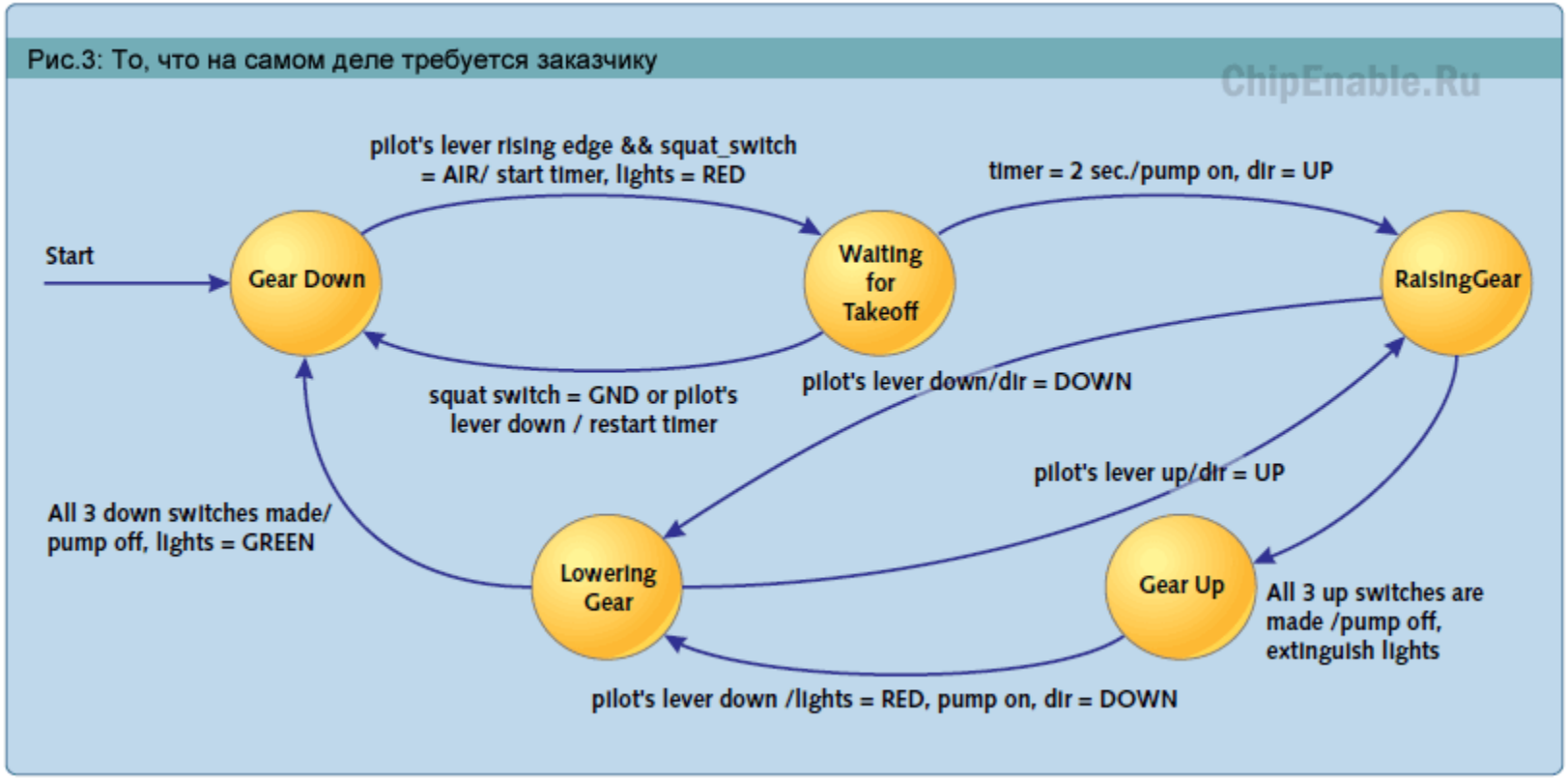
- Рассмотрим грамматику условных операторов:
  S→if E then S| if E then S else S | a
  E→b

- После левой факторизации грамматика принимает вид
  S→ if E then S S' | a
  S' → else S | ε
  E→ b

- К сожалению, грамматика остается неоднозначной.

Программная реализация распознавателя

В качестве распознавателя рассмотрим конечный автомат, представленный графом переходов



Программная реализация распознавателя

Листинг 1. Реализация конечного автомата

```
/*листинг 1*/
typedef enum {GEAR_DOWN = 0, WTG_FOR_TKOFF, RAISING_GEAR, GEAR_UP, LOWERING_GEAR} State_Type;

/*Этот массив содержит указатели на функции, вызываемые в определенных состояниях*/
void (*state_table[])() = {GearDown, WtgForTakeoff, RaisingGear, GearUp, LoweringGear};

State_Type curr_state;

main()
{
  InitializeLdgGearSM();
  /*Сердце автомата – этот бесконечный цикл. Функция, соответствующая
  текущему состоянию, вызывается один раз в итерацию */
  while (1) {
    state_table[curr_state]();
    DecrementTimer();
    /*Здесь можно вызывать другие функции, не связанные с нашим автоматом*/
  }
};
```

Программная реализация распознавателя

Листинг 1. Реализация конечного автомата

```
void InitializeLdgGearSM(void)
{
   curr_state = GEAR_DOWN;
   timer = 0.0;
   /*Остановка аппаратуры, выключение лампочек и т.д.*/
}

void GearDown(void)
{
```

Программная реализация распознавателя

Листинг 1. Реализация конечного автомата

```
    /* Переходим в состояние ожидания, если самолет
     не на земле и поступила команда поднять шасси*/
      if ((gear_lever == UP) && (prev_gear_lever == DOWN) && (squat_switch == UP)) {
         timer = 2.0;
         curr_state = WTG_FOR_TKOFF;
      };
      prev_gear_lever = gear_lever;
   }
```

Программная реализация распознавателя

Листинг 1. Реализация конечного автомата

```
void RaisingGear(void)
{
  /*После того, как все переключатели подняты, переходим в состояние «шасси поднято»*/
  if ((nosegear_is_up == MADE) && (leftgear_is_up == MADE) && (rtgear_is_up == MADE)) {
    curr_state = GEAR_UP;
  };

  /*Если пилот изменил свое решение, перейти в состояние «опускание шасси»*/
  if (gear_lever == DOWN) {
    curr_state = LOWERING_GEAR;
  };
}
```

Программная реализация распознавателя

Листинг 1. Реализация конечного автомата

```
void GearUp(void)
{
  /*если пилот передвинул рычаг в положение «вниз»,
  переходим в состояние «опускание шасси»*/
  if (gear_lever == DOWN) {
    curr_state = LOWERING_GEAR;
  };
}
```

Программная реализация распознавателя

Листинг 1. Реализация конечного автомата

```
void WtgForTakeoff(void)
{
   /* Ожидание перед поднятием шасси.*/
   if (timer <= 0.0) {
      curr_state = RAISING_GEAR;
   };
```

Программная реализация распознавателя

Листинг 1. Реализация конечного автомата

```
    /*Если мы снова коснулись или пилот поменял решение – начать все заново*/
     if ((squat_switch == DOWN) || (gear_lever == DOWN)) {
        timer = 2.0;
        curr_state = GEAR_DOWN;

         /* Don't want to require that he toggle the lever again
        this was just a bounce.*/
        prev_gear_lever = DOWN;
     };
   }

```

Программная реализация распознавателя

Листинг 1. Реализация конечного автомата

```
void LoweringGear(void)
{
   if (gear_lever == UP) {
      curr_state = RAISING_GEAR;
   };

   if ((nosegear_is_down == MADE) && (leftgear_is_down == MADE) &&(rtgear_is_down == MADE)) {
      curr_state = GEAR_DOWN;
   };
}
```

Программная реализация распознавателя

Во-первых, вы можете заметить, что функциональность каждого состояния реализуется отдельной Си функцией.

Код одного состояния никогда не попадет в код другого, если для каждого состояния у вас будет отдельная функция.

Программная реализация распознавателя

Во-вторых, чтобы избежать применения оператора switch, я использую массив указателей на функции состояний, а переменную, используемую в качестве индекса массива, объявляю типа enum.

Спасибо за внимание!!!!!