***LL(k) - Грамматики.***

**Определение LL(k)-грамматик.**

Для начала предположим, что **G**=(**N**,**E**,**P**,**S**) - однозначная грамматика и w=a1,a2...an - цепочка из **L**(**G**). Тогда существует единственная последовательность левовыводимых цепочек b0,b1..bm, для которой **S**=b0,bi,pi  bi+1 при 0<=i<m и am=w. Последовательность p0p1..pm-1 - левый разбор цепочки w.

Допустим, что мы хотим найти этот левый разбор, просматривая w один раз слева направо. Можно попытаться сделать это, строя последовательность левовыводимых цепочек b0,b1..bm. Если bi=a1,a2...ajAB, то к данному моменту анализа мы уже прочли первые j входных символов и сравнили их с первыми j символами цепочки bi. Было бы желательно определить bi+1, зная только a1,a2...aj (часть входной цепочки, считанную к данному моменту), несколько следующих входных символов (aj+1aj+2...aj+k для некоторого фиксированного k) и нетерминал A. Если эти три фактора однозначно определяют, какое правило надо применить для развертки нетерминала A, то ai+1 точно определяется по ai и k входным символам aj+1aj+2...aj+k .

Грамматика, в которой каждый левый вывод обладает этим свойством, называется **LL**(k)-грамматикой. Мы увидим, что для каждой **LL**(k)- грамматики можно построить детерминированный левый анализатор, работающий линейное время. Дадим несколько определений :

**ОПР:** Пусть a=xb такая левовыводимая цепочка в грамматике **G**=(**N**,**E**,**P**,**S**), что xE\*, а b либо начинается нетерминалом, либо пустая цепочка. Будем называть x законченной частью цепочки a, а b - незаконченной частью частью. Границу между x и b будем называть рубежом.

**ПРМ:** Пусть x=abacAaB, тогда abac - законченная часть цепочки x, AaB - незаконченная часть цепочки. Если x=abc, то abc - законченная часть и *е* - незаконченная и рубежом служит конец цепочки.

Иными словами идею **LL**(k) - грамматики можно объяснить так: если имеется уже разобранная часть цепочки, то на основании этого и еще нескольких неразобранных символов мы можем сделать вывод о том, какое правило неоюходимо применить. Таким образом грамматика посуществу не зависит (не считая k последующих символов) от того, что выводится из незаконченной части цепочки. В терминах деревьев этот процесс выглядит следующим образом: дерево вывода цепочки строится начиная с корня и детерминировано сверху вниз.

Вводят функцию FIRST(x) - возвращающую первых k символов. Обычно приписывают в качестве индексов k и **G** - количество символов и грамматика соответственно, но их возможно опускать, если это не вызовет недоразумений.

**ОПР:** KC- грамматика **G**=(**N**,**E**,**P**,**S**) называется **LL**(k)-грамматикой для некоторого фиксированного k, если из существования двух левых выводов

(1) **S**w**Aa`****wb`a`**wx

1. **S**w**Aa`****wc`a`**wy

для которых FIRST(**x**)=FIRST(**y**), вытекает что **b`**=**c`**.

Иначе это определение выражает то, что для имеющейся цепочки и зная следующие k символов можно применить не более одного правила вывода. Грамматика называется **LL**- грамматикой, если она **LL**(k)- грамматика для некоторого k.

**ПРМ:** Пусть **G** состоит из правил **S****aAS**|**b**, **A****a**|**bSA**. Интуитивно **G** является **LL**(1)- грамматикой, потому что, коль скоро дан самый левый нетерминал **С** в левовыводимой цепочке и следующий входной символ **с**, существует не более одного правила, применимого к **С** и приводящего к терминальной цепочке, начинающейся символом **с**. Переходя к определению **LL**(1)- грамматики, мы видим, что если **S****wSa`****wb`a`****wx** и **S****wSa`****wc`a`****wy** и цепочки **x** и **y** начинаются одним и тем же символом , то должно быть **b`**=**c`**. В данном случае если **x** и **y** начинаются символом **a**, то в выводе участвовало правило **S****aAS** и **b`**=**c`**=**aAS**. Альтернатива **S****b** здесь невозможна. С другой стороны, если **x** и **y** начинаются с **b**, то должно применяться правило **S****b** и **b`**=**c`**=**b**. Заметим, что случай **x**=**y**=*e* здесь невозможен, так как из **S** в грамматике **G** не выводится *e*.

Когда рассматриваются два вывода **S****wAa`****wc`a`****wy** рассуждение аналогично. Грамматика **G** служит примером так называемой простой **LL**(1)- грамматики (или разделенной грамматики).

**ОПР:** КС-грамматика **G**=(**N**,**E**,**P**,**S**) без *e*-правил называется простой **LL**(k) - грамматикой ( или разделенной грамматикой ), если для каждого **A****N** все его альтернативы начинаются различными терминальными символами.

**Предсказывающие алгоритмы разбора.**

Разбор для **LL**(k)-грамматики очень удобно осуществлять с помощью так называемого *k- предсказывающего* алгоритма разбора. k-предсказывающий алгоритм использует входную ленту, магазин и выходную ленту. Алгоритм пытается проследить вывод цепочки, записанной на его входной ленте. При чтении анализируемой цепочки входная головка может «заглядывать» вперед на очередные k символа. Эти символы называют *аванцепочкой*. Алгоритм имеет *конфигурацию* представляемую тройкой (*x*,X*a*,***n***), где

*x* - неиспользованная часть входной цепочки

X*a* - цепочка в магазине и Х - верхний символ

***n***  - цепочка на выходной ленте

Работой k- предсказывающего алгоритма руководит управляющая таблица, которая задает соответствие между множеством

{(верхний символ магазина)Х(аванцепочка)}

и множеством

{(правая часть правила и его номер)|ошибка|выброс|допуск}.

Алгоритм является корректным для грамматики, если для любой цепочки из этой грамматики алгоритм позволяет получить упорядоченный список правил для ее разбора. Если работой некоего алгоритма руководит какая-то таблица и этот алгоритм оказывается корректным для рассматриваемой грамматики, то таблицу называют корректной.

**ПРМ:**

Пусть дана грамматика с правилами :

1. **S****aAS**
2. **S****b**
3. **A****a**
4. **A****bSA**

Для такой грамматики будет построена таблица:

аванцепочка

**a** **b**  *e*

верхний **S aAS,1 b,2** ошибка

символ **A** **a,3 bSA,4** ошибка

магазина **a** выброс ошибка ошибка

**b** ошибка выброс ошибка

**$** ошибка ошибка допуск

По такой таблице будет проведен анализ:

(**abbab**,**S$,***e*) |-( **abbab**,**aAS$,1**)

|-( **bbab**,**AS$,1**)

|-( **bbab**,**bSAS$,14**)

|-( **bab**,**SAS$,14**)

|-( **bab**,**bAS$,142**)

|-( **ab**,**AS$,142**)

|-( **ab**,**aS$,1423**)

|-( **b**,**S$,1423**)

|-( **b**,**b$,14232**)

|-(*e*,**$,14232**)

k- предсказывающий алгоритм разбора КС-грамматики **G** можно моделировать на детерминированном МП- преобразователе с концевым маркером на входной ленте. Так как входная головка МП- преобразователя может обозреть только один входной символ, аванцепочка должна храниться в конечной памяти управляющего устройства. Остальные детали моделирования очевидны.

**ТРМ:** Пусть **А** - k- предсказывающий алгоритм разбора для КС-грамматики **G**. Тогда существует такой детерминированный МП- преобразователь, который позволяет разобрать любую цепочку из этой грамматики. Иначе говоря можно промоделировать любой алгоритм на указанном преобразователе.

**СЛВ:** Пусть **А** - k- предсказывающий алгоритм разбора для КС-грамматики **G**. Тогда для **G** существует детерминированный левый анализатор.

**Следствия определения LL(k)-грамматики.**

Покажем что для каждой **LL(k)** грамматики можно механически построить корректный k- предсказывающий алгоритм разбора. Так как ядром алгоритма является управляющая таблица, надо показать, как строить по грамматике такую таблицу. Для этого выведем некоторые следствия определения **LL**(k)- грамматики.

В определении **LL**(k)- грамматики утверждается, что для данной выводимой цепочки wA**a** цепочка w и непосредственно следующие за ней k входных символов однозначно определяют, какое применить правило для развертки нетерминала A. Поэтому на первый взгляд может показаться, что для определения нужного правила надо помнить всю цепочку w. Однако это не так. Докажем теорему, очень важную для понимания **LL**(k)-грамматик:

**ТРМ:** КС-грамматика **G**=(**N**,**E**,**P**,**S**) является **LL**(k)-грамматикой тогда и только тогда, когда для двух различных правил **A****b`** и **A****c`** из **P** пересечение FIRST(**b`a`**)FIRST(**c`a`**) пусто для всех таких wA**a`**, что **S**wA**a`**.

**ДКВ:** Необходимость. Допустим, что w, A, **a`**, **b`** и **c`** удовлетворяют условиям теоремы, а FIRST(**b`a`**)FIRST(**c`a`**) содержит x. Тогда по определению FIRST для некоторых y и z найдутся выводы **S**wA**a`**w**b`a`**wxy и **S**wA**a`**w**c`a`**wxz. (Заметим, что здесь мы использовали тот факт, что **N** не содержит бесполезных терминалов, как это предполагается для всех рассматриваемых грамматик.) Если |x| < k то y = z = *e*. Так как **b`**  **c`**, то **G** не **LL**(k)- грамматика.

Достаточность. Допустим, что G не LL(k)- грамматика. Тогда найдутся такие два вывода **S**wA**a`**w**b`a`**wx и **S**wA**a`**w**c`a`**wy, что цепочки x и y совпадают в первых k позициях, но **b`****c`**. Поэтому A**b`** и A**c`** - различные правила из P и каждое из множеств FIRST(**b`a`**) и FIRST(**c`a`**) содержит цепочку FIRST(x) совпадающую с FIRST(y). **ЧТД**.

**ПРМ:** Грамматика **G** из правила **S**a**S**|a, не будет **LL**(1)- грамматикой, так как FIRST1(a**S**)=FIRST1(a)=a. Это можно объяснить так - видя только первый символ цепочки мы не можем определить какое правило необходимо применить (левое или правое). С другой стороны эта грамматика является LL2(k) грамматикой - что вполне очевидно.

**ОПР:** Пусть **G**=(**N**,**E**,**P**,**S**) - КС-грамматика. Определим FOLLOWk(**b`**) как множество терминальных символов, которые могут встречаться после нетеминала, являющегося аргументом функции.

**ТРМ:** КС-грамматика **G**=(**N**,**E**,**P**,**S**) является **LL**(1)-грамматикой тогда и только тогда, когда для двух различных правил **A****b`** и **A****c`** пересечение FIRST1(**b`** FOLLOW1(A))FIRST1(**c`** FOLLOW1(A)) пусто при всех A**N**. (Без **ДКВ**).

Теорему можно выразить следующим : по первому символу после нетерминала необходимо выбрать применимое правило - следовательно эти символы различны и пересечение пусто. Эта теорема может применяться к **LL**(k)- грамматикам, но не всегда выполняться. Грамматики для которых выполняется теорема называются сильными, таким образом все **LL**(1) грамматики - сильные. Необходимо так же заметить что каждая **LL**(k)- грамматика однозначна, поэтому если имеется неоднозначная грамматика - то она не **LL**(k). Имеется неразрешимая проблема распознавания, существует ли для данной КС-грамматики **G**, не являющейся **LL**(k), эквивалентная ей **LL**(k)- грамматика. Однако в ряде случаев такое преобразование возможно. Применяется два способа:

Первый способ - *устранение левой рекурсии*.

**ПРМ:** Пусть **G** - грамматика **S****S**a|b которая не является LL- грамматикой. Заменим правила на следующие:

**S** b**S`**

**S`**a**S`**|*e*

получив при этом эквивалентную **LL**(1)- грамматику.

Другой способ - *левая факторизация*.

**ПРМ:** Рассмотрим **LL**(2)- грамматику **G** с двумя правилами **S**a**S**|a. В этих двух правилах «вынесем влево за скобки» символ a, записав их в виде **S**a(**S**|*e*). Иными словами, мы считаем что операция конкатенации дистрибутивна относительно операции выбора альтернативы (обозначаемой вертикальной чертой). Заменим эти правила на :

**S**aA

A**S**|*e*

тем самым получим эквивалентную **LL**(1)-грамматику.

**Разбор для LL(1)- грамматик.**

Ядром предсказывающего алгоритма является управляющая таблица. В этом и следующих разделах будет показано как построить корректную управляющую таблицу.

**АЛГ 1:** Управляющая таблица для **LL**(1)-грамматики.

*Вход :* **LL**(1)- грамматика .

*Выход :* Корректная управляющая таблица.

*Метод :* Будем считать, что $-маркер дна магазина. Таблица определяется следующим образом:

1. Если A  **a`** - правило из P с номером i, то M[A,a]=(**a`**,i) для всех ae, принадлежащих FIRST1(**a`**). Если *e*FIRST1(**a`**), то M[A,b]=(**a`**,i) для всех bFOLLOW1(A).
2. M[a,a]=выброс для всех aE.
3. M[$,*e*]=допуск.
4. В остальных случаях M[X,a]=ошибка для XNE{$} и aE{e}.

**ТРМ:** Предложенный алгоритм строит корректную управляющую таблицу для **LL**(1)- грамматики **G**.

**Разбор для LL(k)- грамматик.**

Построим управляющую таблицу для произвольной грамматики. Если грамматика сильная, то можно применить предыдущий алгоритм с аванцепочками расширенными до k символов. В противном случае возникает несколько проблем. В 1-м предсказывающем алгоритме разбора в магазин помещались только символы из **E****N** и оказывалось, что для однозначного определения очередного применяемого правила достаточно знать нетерминальный символ наверху магазина и текущий входной символ. Для не сильных грамматик этого может оказаться недостаточно.

**ПРМ:** Возьмем грамматику

**S**aAaa|bAba

Ab|*e*

Если даны нетерминал A и аванцепочка ba, то не известно, какое из правил надо применить.

Неопределенности такого рода однако можно разрешить, связав с каждым нетерминалом и частью левовыводимой цепочки (которая может оказаться справа), специальный символ, называемый **LL**(k)- *таблицей*. По данной аванцепочке **LL**(k)- таблица будет однозначно определять какое правило надо применить на очередном шаге вывода.

**ОПР:** Пусть **E** - некоторый алфавит. Если **L**1 и **L**2 - подмножества **E**, то положим **L**1k **L**2 = {

w | для некоторых x**L**1 и y**L**2

либо w = xy, если |xy|k

либо w состоит из первых k символов цепочки xy

}

**ЛМА:** Для любой КС- грамматики **G**=(**N**,**E**,**P**,**S**) и любых **a`**, **b`**(**N****E**) :

FIRSTk(**a`b`**)=FIRSTk(**a`**)k FIRSTk(**b`**)

**ОПР:** Пусть **G**=(**N**,**E**,**P**,**S**) - КС- грамматика. Для каждых A**N** и L**E** определим **LL**(k)- таблицу Ta,l, соответствующую A и L, как функцию T(u), значением которой служит :

1. =ошибка, если в P нет такого правила A**a`**, что FIRSTk(**a`**) k L содержит u;
2. =(A**a`**,<Y1,Y2...Ym>), если A**a`** - единственное правило из P, для которого FIRSTk(**a`**) k L содержит u;
3. не определено, если в множестве найдутся два и более правила (эту ситуацию называют конфликтом между правилами)

На нормальном языке это означает что вырабатывается значение ошибка, если u вообще не является цепочкой грамматики, возвращается правило если оно существует и только одно и если несколько правил - то значение не определяется.

**АЛГ 2:** Построение **LL**(k)- таблиц.

*Вход:* **LL**(k)- грамматика **G**=(**N**,**E**,**P**,**S**).

*Выход:* Множество TG **LL**(k)- таблиц, необходимых для построения управляющей таблицы для **G**.

*Метод:*

1. Построить **LL**(k)- таблицу T0, соответствующую **S** и {*e*}.
2. Положить вначале TG={T0}.
3. Для каждой LL(k)-таблицы TTG, содержащей элемент T(u)=(Ax0B1x1...Bmxm,<Y1,Y2...Ym>) включить в TG **LL**(k)- таблицу T для 1im, если T еще не входит в TG.
4. Повторять шаг 3 пока в TG можно включать новые таблицы.

**ПРМ:** Построим соответствующее множество **LL**(2)- таблиц для грамматики **S**aAaa|bAba и Ab|*e*. Начнем с TG={TS,{*e*}} . Так как TS,{*e*}(aa)=( **S**aAaa,{aa}), то в TG надо включить TA,{aa}. Аналогично, так как T0(bb)=( **S**bAba,{ba}), то в TG нужно так же включить . (Элементы **LL**(2)- таблиц TA,{aa} и TA,{ba}, отличные от значения ошибка, приведены в таблице ниже). Сейчас TG={TS,{e},TA,{aa},TA,{ba}}, и алгоритм уже не может включить в TG новых таблиц, так что это три **LL**(2)- таблицы образуют множество соответствующее грамматике.

TA,{aa} TA,{ba}

u правило множества u правило множества

ba A  b - ba A  *e* -

aa A  *e* - aa A  b -

Теперь дадим алгоритм, которым можно построить корректную управляющую таблицу по соответствующему множеству **LL**(k)- таблиц. Управляемый этой таблицей k- предсказывающий алгоритм будет в качестве магазинных символов употреблять вместо нетерминалов **LL**(k)- таблицы.

**АЛГ 3:** Построение управляющей таблицы для **LL**(k)- грамматики.

*Вход :* **LL**(k)- грамматика и соответствующее множество TG **LL**(k)- таблиц.

*Выход :* Корректная управляющая таблица M для **G**.

*Метод:* M определяется на множестве (TGE{$})E\*k следующим образом:

1. Если Ax0B1x1...Bmxm - правило из P с номером i и TA,LTG, то для всех u, для которых TA,L(u)=(Ax0B1x1...Bmxm,<Y1...Ym>) полагаем M[TA,L,u]=(x0TB1,Y1...TBm,Ymxm,i).
2. M[a,av]=выброс для всех vE\*(k-1).
3. M[$,*e*]=допуск.
4. В остальных случаях M[X,u]=ошибка
5. TS,{*e*} - начальная таблица.

**ПРМ:** Построим управляющую таблицу для **LL**(2)- грамматики

1. **S**aAaa
2. **S**bAba
3. Ab
4. A*e*

используя соответствующее ей множество **LL**(2)-таблиц, найденное в предыдущем примере. Алгоритм должен выдать таблицу:

aa ab a ba bb b *e*

T0 aT1aa,1 aT1aa,1 bT2ba,2

T1 *e*,4 b,3

T2 *e*,4 b,3

a выброс выброс выброс

b выброс выброс выброс

$ допуск\*

где T0=TS,{e}, T1=TA,{aa} и T2=TA,{ba}. Подразумевается, что в пустых ячейках - ошибка. Допуск\* находится в последней колонке. Для входной цепочки bba 2-предсказывающий алгоритм выдаст такую последовательность тактов:

(bba,T0$,*e*) |- (bba,bT2ba$,2)

|- (ba,T2ba$,2*)*

|- (ba,ba$,24)

|- (a,a$,24)

|- (*e*,$,24)

**ТРМ:** Описанный алгоритм строит для **LL**(k)- грамматики **G**=(**N**,**E**,**P**,**S**) корректную таблицу, управляющую работой соответствующего k- предсказывающего алгоритма.

**ПРМ:** Рассмотрим **LL**(2)- грамматику G с правилами:

1. **S***e*
2. **S**abA
3. A**S**aa
4. Ab

Построим соответствующие **LL**(2)-таблицы. Начнем с T0=TS,{*e*}. Затем по T0 найдем T1=TA,{*e*}, далее T2=TS,{aa} и T3=TA,{aa}:

T0 T2

u правило множества u правило множества

*e* **S**e - aa **S***e* -

ab **S**abA {*e*} ab **S**abA {aa}

T1 T3

u правило множества u правило множества

b Ab - aa A**S**aa {aa}

aa A**S**aa {aa} ab A**S**aa {aa}

ab A**S**aa {aa} ba Ab -

По этим таблицам построим управляющую таблицу:

aa ab a ba bb b *e*

T0 abT1,2 *e*,1

T1 T2aa,3 T2aa,3 b,4

T2 *e*,1 abT3,2

T3 T2aa,3 T2aa,3 b,4

a выброс выброс выброс

b выброс выброс выброс

$ допуск

Алгоритм построенный по таблице разберет цепочку abaa следующим образом:

(abaa,T0$,*e*) |- (abaa,abT1$,2)

|- (baa,bT1$,2)

|- (aa,T1$,2)

|- (aa,T2aa$,23)

|- (aa,aa$,231)

|- (a,a$,231)

|- (*e*,$,231)

**ТРМ:** Число шагов, выполняемых k- предсказывающим алгоритмом с управляющей таблицей, построенной предыдущим алгоритмом по **LL**(k)- грамматике G, линейно зависит от n, где n - длинна входной цепочки.

**Проверка LL(k)- условия.**

По отношению к произвольной данной грамматике G возникает ряд естественных вопросов:

1. Является ли **G** **LL**(k)- грамматикой для данного k ?
2. Существует ли такое k, что **G** - **LL**(k)- грамматика?
3. Так как для **LL**(1) левые разборы строятся особенно просто, то если G не **LL**(1)- грамматика, то существует ли эквивалентная ей **LL**(1)- грамматика **G’**, для которой L(**G**) = L(**G’**)?

К сожалению, только для первого вопроса есть отвечающий на него алгоритм. Можно показать, что вторая и третья проблемы алгоритмически не разрешимы, но это доказательство не приводится. Приведем алгоритм проверки **LL**(k)- условия:

**АЛГ 4:** Проверка **LL**(k)- условия.

*Вход:* КС- грамматика **G**=(**N**,**E**,**P**,**S**) и целое число k.

*Выход:* «Да» - если **G** - **LL**(k)- грамматика и «Нет» в противном случае.

*Метод:*

Суть алгоритма сводится к следующему: Для каждого нетерминала, имеющего два или более правила раскрутки вычисляется пересечение первых k- символов всех возможных цепочек раскрутки. Если это множество пусто, то переходят к следующему терминалу, иначе заканчивают со значением «Нет». Если все пересечения пусты - заканчивают со значением «Да». Для получения пересечения двух правил можно воспользоваться записью: (FIRSTk(**b`**)kL)(FIRSTk(**c`**)kL), где L=FIRSTk(**a`**) и **a`** - цепочка символов после терминала.