**Контекстно-свободные грамматики**

***КС-грамматикой*** (или ***контекстно-свободной грамматикой*** и ***грамматикой типа 2***) называется грамматика с правилами вида *A* **, где *A* *N*, **  *V\**. Язык, порождаемый КС-грамматикой, называется ***КС-языком*** (***языком*** ***типа 2***).

Напомним определения

Определение. Пусть G – грамматика. Говорят, что цепочка v *непосредственно порождает* цепочку w, и обозначают это

v  w ,

если для некоторых цепочек x и y можно написать

v=x U y, w=x u y,

где U ::= u – правило грамматики G. Также говорят, что w непосредственно выводима из v или что w непосредственно приводится к v.

Например, цепочка <цифра> 1, непосредственно порождает цепочку - 51. Цепочки x и y могут быть пустыми. Следовательно, для любого правила U ::= u грамматики G имеет место U  u.

Определение. Говорят*, v порождает w* или *w приводится к v*, что записывается как v +w, если существует последовательность непосредственных выводов v u0 u1 u2 …  u[n]=w, где n>0. Эта последовательность называется *выводом длины n*. Говорят также, что цепочка w является *словом* для v. И пишут v \*w, если v +w или v=w.

Например, цепочка <число> 1, порождает цепочку - 51.

Определение. Пусть G[Z]- грамматика. Цепочка x называется *сентенциальной формой*, если x выводима из начального символа Z, то есть если Z \*x. *Предложение* – это сентенциальная форма, состоящая только из терминальных символов. *Язык* L(G[Z]) – это множество предложений:

Например, цепочки <число> 1, <цифра> 1,51- *сентенциальные формы.*

51-предложение языка целых чисел.

Таким образом*, язык* – это просто подмножество множества всех терминальных цепочек, то есть цепочек в VT. Структура предложения задается грамматикой. Несколько различных грамматик могут порождать один и тот же язык.

Определение. Пусть G[Z]- грамматика. И пусть w= xuy – сентенциальная форма. Тогда u называется *фразой* сентенциальной формы w для нетерминального символа U, если Z \*xUy и U +u ; и далее, u называется *простой фразой*, если Z \*xUy и U u.

Например, для *сентенциальной формы* <цифра> 1 *.* <цифра> -простая фраза, т.к. существует правило <число> ::= <цифра>

Тот факт, что и U +u, вовсе не означает, что u является фразой сентенциальной формы xuy; необходимо также иметь Z \*xUy.

Определение. *Основой* всякой сентенциальной формы называется самая левая простая фраза.

Грамматика G1 описывает бесконечный язык, то есть язык, состоящий из бесконечного числа предложений. Это обусловлено тем, что правило <чс>::=<чс><цифра> содержит <чс> и в левой, и в правой частях, то есть в некотором смысле символ <чс> сам себя определяет.

Примером является грамматика языка {*anbn* | *n* 0}:

**Пример** 3.4. Язык {*anbn* | *n* 0} порождается КС-грамматикой с правилами

*S* *aSb*,    *S* *.*

Это вытекает из того, что все выводы в этой грамматике имеют вид

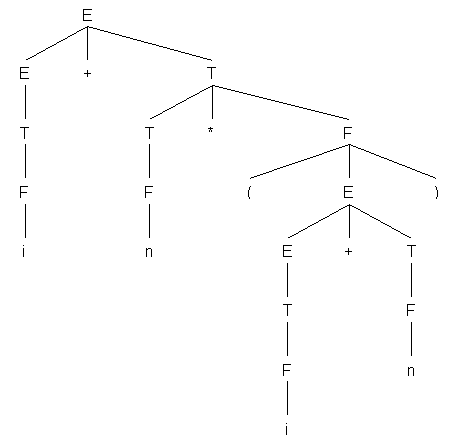
*S* *aSb* *aaSbb* ... *anSbn* *anbn.*  


Поскольку вывод из нетерминального символа теперь не зависит от контекста, правомерно также для каждого *A* *N* рассматривать порождаемый им язык *LG*(*A*) = {**| ***T*\*, *A*\***}.

Каждому выводу в КС-грамматике можно сопоставить дерево вывода. ***Деревом вывода в грамматике*** *G* называется дерево следующего вида:

* корнем является нетерминальный символ;
* вершинами являются символы из *V*;
* если вершина *A* имеет непосредственных потомков *X*1, ..., *Xn* (или **), то в грамматике есть правило *A* *X*1...*Xn* (или *A* **). Порядок потомков совпадает с порядком символов в правиле.

**Пример** 3.5. Дерево вывода цепочки *i+n\**(*i+n*) в грамматике арифметических выражений





Вообще говоря, одно дерево может соответствовать нескольким выводам, в зависимости от порядка обхода дерева. Обходу "сверху вниз, слева направо" соответствует так называемый ***левый вывод***, при котором правило применятся всегда к самому левому нетерминальному символу. В предыдущем примере левый вывод имеет вид

*E* *E+T* *T+T* *F+T* *i+T*  *i+T\*F* *i+F\*F* *i+n\*F* *i+n\**(*E*)  *i+n\**(*E+T*) *i+n\**(*T+T*) *i+n\**(*F+T*)**  
*i+n\**(*i+T*)***i+n\**(*i+F*)***i+n\**(*i+n*)

КС-грамматика *G* называется ***однозначной***, если у каждой цепочки языка *L*(*G*) имеется только одно дерево вывода (и, следовательно, один левый вывод). КС-язык называется ***однозначным***, если он порождается какой-нибудь однозначной грамматикой. Предыдущий язык выражений является однозначным.

Легко построить примеры неоднозначных грамматик. Достаточно, например, в произвольную КС-грамматику ввести правило *A* *A* для какого-нибудь нетерминального *A*. Сложнее найти неоднозначный язык. Справедлива следующая теорема (мы приводим её без доказательства).

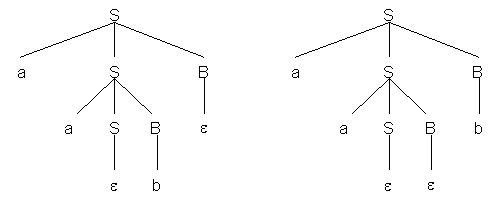
Т е о р е м а  3.4. Проблемы однозначности КС-грамматик и КС-языков алгоритмически неразрешимы.  


Неоднозначные грамматики могут использоваться для описания и трансляции языков программирования.

**Пример** 3.6. Построим модель условного оператора языка Паскаль. Пусть терминальный символ *a* соответствует конструкции if <выражение> then, терминальный символ *b* соответствует конструкции else <оператор>, нетерминальный символ *S* соответствует конструкции <условный оператор>, а нетерминальный символ *B*  конструкции <возможный else>. Тогда типичная грамматика условного оператора (если отвлечься от других операторов) может быть промоделирована КС-грамматикой с правилами:

*S*  *aSB* | **,  *B*  *b* | **

Эта грамматика порождает язык {*anbm* | *n*  *m* 0}, приписывая цепочке *aab* два дерева вывода

  
В Паскале принято соглашение о связывании else с ближайшим слева if, что соответствует дереву, изображённому слева. Этот язык однозначен, так как порождается однозначной грамматикой

*S*  *AX* | **,  *A*  *aA* | **,  *X*  *aXb* | **,

которая к тому же определяет правильную структуру дерева. Но эта грамматика вызывает большие трудности при синтаксическом анализе и поэтому не используется.   


Если в КС-грамматике имеется *n* правил *A* *i* (1 *i*  *n*) для нетерминального символа *A*, то их можно объединять в записи

(4)            *A* **1 | ... | *n*.

Каждой КС-грамматике *G* = (*N*,*T*,*P*,*S*) поставим в соответствие БНФ над алфавитом T и множеством переменных *N*, в которой каждому правилу вида (4) соответствует уравнение

**Приведение грамматик**

Разбор предложений грамматики выполняется проще, если грамматика не содержит лишних правил. Можно с достаточным основанием утверждать, что если грамматика языка программирования содержит лишние правила, то она ошибочна. Следовательно, алгоритм, который обнаруживает лишние правила в грамматике, может оказаться полезным при проектировании языка.

Правило, подобное U ::=U, очевидно, не является необходимым для грамматики; более того, оно приводит к неоднозначности грамматики. Поэтому далее полагается, что грамматика не содержит правил вида

# U ::=U.

# Грамматики могут также содержать лишние правила, которые невозможно использовать в выводе хотя бы одного предложения. Например, в следующей ниже грамматике G[4] нетерминал <d> не может быть использован в выводе какого – либо предложения, так как он не встречается ни в одной из правых частей правил:

Z ::= <b> e

<a> ::= <a> e | e

<b> ::= <c> e | <a> f

<c> ::= <c> f

<d> ::= f

Чтобы появиться в выводе какого – либо предложения, нетерминал U должен удовлетворять двум условиям. Во – первых, символ U должен встречаться в некоторой сентенциальной форме:

(2) Z🡪 \*xUy для некоторых цепочек x и y,

где Z- начальный символ грамматики. Во- вторых, из U должна выводиться цепочка t, состоящая из терминальных символов:

(3) U🡪 +t для некоторой t принадлежащей VT+ .

Ясно, что если нетерминальный символ U не удовлетворяет этим условиям, то правила, содержащие U в левой части, не могут быть использованы ни в каком выводе. С другой стороны, если все нетерминалы удовлетворяют этим условиям, то грамматика не содержит лишних правил.

Так, если U ::=u – правило, то, во –первых, Z🡪 \*xUy 🡪 x u y. Во-вторых, так как мы можем вывести цепочку терминальных символов для каждого нетерминала, содержащего в цепочке x u y, то

x u y 🡪 \* t для некоторой t принадлежащей VT+ .

В конечном итоге мы получаем Z 🡪 +t, то есть вывод, в котором было использовано правило U ::=u.

Условие (2) проверяет нетерминалы на недостижимость. Нетерминалы, которые не появляются ни в одной цепочке, выводимой из начального символа, называются *недостижимыми нетерминалами*.

Условие (3) проверяет нетерминалы на бесплодность. Нетерминалы, которые не порождают ни одной терминальной цепочки, называются *бесплодными* или *мертвыми*.

Нетерминалы, которые бесплодны или недостижимы, называются *лишними (бесполезными).*

В описанной выше грамматике G4 нетерминальный символ <c> не удовлетворяет условию (3), так как при непосредственном выводе он должен быть заменен на <c> f. Такая замена не может привести к цепочке из терминальных символов. Если отбросить все правила, которые нельзя использовать при порождении хотя бы одного предложения, то останутся правила Z ::= <b> e, <a> ::= <a> e, <a> ::= e, <b> ::= <a> f .

Определение. Грамматика G называется *приведенной*, если каждый нетерминал U удовлетворяет условиям (2) и (3), то есть не содержит лишних нетерминалов.

Нетерминал U удовлетворяет условию 2, тогда и только тогда, когда Z WITHIN+ U, где WITHIN+ является транзитивным замыканием отношения WITHIN (внутри). U WITHIN S тогда и только тогда, когда существует правило U::=…S….

Другими словами, если нетерминал в левой части правила является *достижимым*, то достижимы и все символы правой части этого правила. Это свойство выполняется, так как можно сначала вывести цепочку, содержащую символ, который является левой частью правила, и потом применить к ней это правило. Блок схема алгоритма, проверяющего символы на достижимость, приведена на рис.1. Список, полученный в результате работы алгоритма, содержит только достижимые нетерминалы, а все нетерминалы, не попавшие в этот список являются недостижимыми.

На рис.2 приведена блок схема алгоритма, который проверяет нетерминалы, встречающиеся в наборе правил на продуктивность (проверка условия (3)). Терминальный или нетерминальный символ называется *продуктивным*, если из него выводится какая-нибудь терминальная цепочка (то есть он не является бесплодным терминалом). Работа алгоритма начинается с того, что некоторым образом отмечаются те нетерминалы, для которых существует правило U ::= t , где t принадлежит VT+ (первое выполнение блока 1). Такие нетерминалы, очевидно удовлетворяют условию (3). Далее в алгоритме проверяются все неотмеченные нетерминалы U и для каждого из них делается попытка найти правило U ::= x, где x состоит только из терминальных символов или ранее отмеченных нетерминалов (повторное выполнение блока 1). Символы, для которых такое правило существует, также, очевидно удовлетворяют условию (3). Этот процесс продолжается до тех пор, пока либо все нетерминалы не будут отмечены, либо пока при выполнении блока 1 не будет отмечено ни одного символа. Работа алгоритма завершена. Неотмеченные нетерминалы не удовлетворяют условию (3).

**Практические задания.**

1. Исключите лишние правила:

а) Z ::= E+T T ::= T+i ¦ i

E ::= E ¦ S+F ¦ T G ::= G ¦ GG ¦ F

F ::= F ¦ FP ¦ P Q ::= E ¦ E+F ¦ T ¦ S

P ::= G S ::= i

б)S⇒A ¦ B в) S⇒a¦A

A ⇒ aB ¦ bS ¦ b A⇒AB

B ⇒ AB ¦ Ba B⇒ b

B⇒ AS ¦ b

г) S⇒ A ¦ B

A⇒ C ¦ D

B⇒ DE

C⇒ S ¦ a ¦ e

D⇒ S ¦ b

E ⇒ S ¦ C ¦ e

д) S ⇒ bCES е) S ⇒ aABC ¦ aE

A ⇒ bE ¦ d A ⇒ aE ¦ SCD

B ⇒ aBC B ⇒ dFS

C ⇒ bE C ⇒ aES

D ⇒ d D ⇒ aAC

E⇒ e E ⇒ aCE

F⇒ aF F ⇒ AB

**Лемма подкачки. МП-автомат для КС грамматики**

        Если в KC-грамматике G существует нетерминал A, такой, что A       |      0 , то говорят, что грамматика G содержит *самовложение*. Символ   обозначает существование вывода одной цепочки из другой (\* означает нуль или более применений отношения  ).

*Примеры грамматик, содержащих самовложения:*

(G1) S aSb

S 

(G2) S begin A end

S  A [ S ]

        Теоретически любая КС-грамматика, не содержащая самовложений эквивалентна регулярной грамматике. Именно самовложения позволяют отличать регулярные языки от нерегулярных.

***Лемма подкачки (Огдена)***

        Для любого КС-языка существует такая константа k, что |Z|>k, где Z-предложение языка L, и можно записать Z=uυωxy, γδе |υωx| k и |υ| и |x|  0, то строка uυiωxiy (i  0) L.   
С помощью этой леммы можно показать, что определенные языки не являются КС языками:

{LL| L {0, 1}\*} для него существует константа k.

Пусть L состоит из k нулей и k единичек {0…0, 1…1}, тогда Z = LL = 0…0,1…1,0…0,1…1.

Z = uυωxy. .κ. | Z|>k , a | υ x |<k

Возможны два варианта:

1. υ θ x νаходятся в первой половине Z или во второй, т.е. состоят из нулей или единиц.
2. υ θ x ρодержат единички из первой половины строки.

        Рассмотрим первый случай. Так как длина υ x  k и υ и x  0, то uωy (i=0) образуется исключением из Z 0 и 1 первой половины, или второй, но не из обеих половин. Таким образом, uωy  L.   
        Рассмотрим второй случай. uωy (i=0) исключает из Z единицы из первой части(L) и нули из второй, т.е. из середины слова Z. Полученное слово не принадлежит языку.