Приведенные грамматики

*Приведенные грамматики —* это КС-грамматики, которые не содержат недостижимых и бесплодных символов, циклов и ε-правил («пустых» правил). Приве­денные грамматики называют также КС-грамматиками в каноническом виде.

Для того чтобы преобразовать произвольную КС-грамматику к приведенному виду*,* необходимо выполнить следующие действия:

* удалить все бесплодные символы;
* удалить все недостижимые символы;
* удалить ε-правила;
* удалить цепные правила.

Следует подчеркнуть, что шаги преобразования должны выполняться именно в указанном порядке, и никак иначе.

Приведение грамматики – удаление правил с бесполезными символами – может быть выполнено на основании двух очевидных свойств. Если все нетерминальные символы в правой части правила – производящие, то и символ в левой части правила – производящий. Производящий нетерминал N строго может быть определен следующим образом: N ⇒+ μ, μ ∈ VT+.

Если символ левой части достижим, то и символы правой части достижимы. Иначе говоря, для любого нетерминала должно быть справедливо: S ⇒\* αNβ, где α, β∈V\*.

## Преобразования грамматик

В некоторых случаях КС-грамматика может содержать недостижимые и бесплодные символы, которые не участвуют в порождении цепочек языка и поэтому могут быть удалены из грамматики.

**Определение:** символ A  **VN** называется *бесплодным* в грамматике G = (**VT**, **VN**, **P**, S), если множество { α | α  **VT\***, A  α} пусто.

**Неформально:**Последовательно просматриваются правила грамматики. Нетерминальный символ из левой части правила считается порождающим, если цепочка в правой части правила содержит только терминальные символы, или все нетерминалы этой цепочки являются элементами множества производящих символов. Последовательные просмотры правил продолжаются до тех пор, пока после очередного просмотра в множество производящих символов не будет добавлено ни одного нового элемента. Все правила, содержащие нетерминалы, не попавшие в множество производящих символов, удаляются из грамматики.

**Формально:**

**Алгоритм удаления бесплодных 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 N0, N1, ...

1. **N0** = ∅, i = 1.
2. **Ni** = {A | (A  α)  **P** и α  (**Ni-1**  **VT**)\*}  **Ni-1**.
3. Если **Ni**  **Ni-1**, то i = i+1 и переходим к шагу 2, иначе **VN’** = **Ni**; **P’** состоит из правил множества **P**, содержащих только символы из **VN’** ∪ **VT**; G’ = (**VT**, **VN’**, **P’**, S).

**Определение:** символ x  (**VT**  **VN**) называется *недостижимым* в грамматике G = (**VT**, **VN**, **P**, S), если он не появляется ни в одной сентенциальной форме этой грамматики.

**Неформально:**

Символ S считается достижимым. Далее последовательно просматриваются правила в левой части которых находятся достижимые нетерминалы и все символы из правой части этих правил помещаются в множество достижимых. Последовательные просмотры правил продолжаются до тех пор, пока после очередного просмотра в множество достижимых символов не будет добавлено ни одного нового элемента. Все правила, содержащие недостижимые символы удаляются из грамматики.

**Формально:**

**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. V0 = {S}; i = 1.
2. Vi = {x | x  (VT  VN), (A  αxβ) ∈ P и A  Vi-1}  Vi-1.
3. Если Vi  Vi-1, то i = i+1 и переходим к шагу 2, иначе VN’ =   
   Vi ∩ VN; VT’ = Vi ∩ VT; P’ состоит из правил множества P, содержащих только символы из Vi; G’ = (VT’, VN’, P’, S).

**Определение:** КС-грамматика G называется *приведенной,* если в ней нет недостижимых и бесплодных символов.

**Алгоритм приведения грамматики**:

1. обнаруживаются и удаляются все бесплодные нетерминалы.
2. обнаруживаются и удаляются все недостижимые символы.

Удаление символов сопровождается удалением правил вывода, содержащих эти символы.

**Замечание: е**сли в этом алгоритме переставить шаги (1) и (2), то не всегда результатом будет приведенная грамматика.

Для описания синтаксиса языков программирования стараются использовать однозначные приведенные КС-грамматики.

Преобразование неукорачивающих грамматик

Последний вид рассматриваемых преобразований связан с удалением из грамматики правил с пустой правой частью.

***Определение****.* Правило вида A **** ε называется **«пустым»** (**аннулирующим) правилом.**

***Определение****.* Грамматика называется **неукорачивающей** или грамматикой без «пустых» правил, если либо

1)схема грамматики не содержит аннулирующих правил,

2)либо схема грамматики содержит только одно правило вида S **** ε, где S - начальный символ грамматики, и символ S не встречается в правых частях остальных правил грамматики.

Для грамматик, содержащих аннулирующие правила, справедливо следующее утверждение.

***Утверждение***. Для каждой КС-грамматики G', содержащей аннулирующие правила, можно построить эквивалентную ей неукорачивающую грамматику G, такую что L(G')=L(G).

Построение неукорачивающей грамматики приведет к увеличению числа правил заданной грамматики из-за построения дополнительных правил, получаемых в результате исключения нетерминалов аннулирующих правил. Чтобы построить дополнительные правила необходимо выполнить все возможные подстановки пустой цепочки вместо аннулирующего нетерминала во все правила грамматики.

Если же в грамматике есть правило вида S **** ε, где S – начальный символ грамматики, и символ S входит в правые части других правил грамматики, то следует ввести новый начальный символ S’ и заменить правило S **** ε двумя новыми правилами: S' **** ε и S'**** S.

В качестве иллюстрации способа построения неукорачивающих грамматик, исключим аннулирующие правила из следующей грамматики:

G ({a,b}, {S}, P = { S  aSbS, S  bSaS, S  ε }, S).

Выполняя все возможные замены символа S в первом правиле грамматики, получаем четыре правила вида:

S  aSbS, S  abS, S  aSb, S  ab .

Поступая аналогично со вторым правилом, имеем:

S bSaS, S baS, S  bSa, S  ba.

Учитывая, что начальный символ, образующий аннулирующее правило, входит в правые части других правил грамматики, заменим правило S **** ε правилами вида S' **** ε и S' **** S.

Построенная совокупность правил образует множество правил искомой неукорачивающей грамматики.

S' **** S | ε

S  aSbS | abS | aSb | ab | bSaS |baS | bSa | ba

Исключение цепных правил

***Определение*.** Правило грамматики вида A **** B, где A,B **∈VN**, называется **цепным.**

***Утверждение****.* ДляКС-грамматики **G**, содержащей цепные правила , можно построить эквивалентную ей грамматику **G**', не содержащую цепных правил.

Идея доказательства заключается в следующем.

Если грамматика G имеет правила A **** B, B **** C, C ****aX, то такие правила могут быть заменены одним правилом А **** aX, поскольку вывод A **** B **** C ****aX цепочки aX в грамматике G может быть получен в грамматике G' с помощью правила A **** aX.

В общем случае доказательство последнего утверждения можно выполнить так.

Разобьем множество правил P грамматики G на два подмножества P1 и P2, включая в P1 все правила вида A ****B.

Для каждого правила из P1 найдем множество правил S(Ai), которые строятся так: если Ai **** \* Aj и в P2 есть правило Aj ****  , где  - цепочка словаря **(VN VT)**\*, то в S(Ai) включим правило Ai ****  .

Построим новое множество правил P’ путем объединения правил P2 и всех построенных множеств S(Ai). Получим грамматику G' = {VN,VT, P’, S}, которая эквивалентна заданной и не содержит правил вида A **** B.

В качестве примера выполним исключение цепных правил из грамматики **G** :

G = ({+,\*,(,),a}, {E,T,F}, P={E  E+T | T, T  T\*F | F, F  (E) | a}, E).

Вначале разобьем правила грамматики на два подмножества:

P1 = {E **** T, T **** F} ,

P2 = {E **** E+T, T **** T\*F, F ** **(E) | a }

Для каждого правила из P1 построим соответствующее подмножество.

S(E) = { E **** T\*F, E ****(E) | a },

S(T) = { T **** (E) | a}

В результате получаем искомое множество правил грамматики без цепных правил в виде:

P2 U S(E) U S(T) = { E **** T+T | T\*F | (E) | a, T **** T\*F | (E) | a, F **** (E) | a }

Все приведенные выше преобразования грамматик могут быть использованы при построении как конечных, так и магазинных автоматов.