1.1 Упрощение КС грамматик

Прежде чем разрабатывать распознаватель для КС языков, заданных КС-грамматикой, имеет смысл упростить эту грамматику, которая, например, могла быть получена формальным способом, удалив из нее все правила, которые невозможно использовать в выводе хотя бы одного предложения.

Пусть $G = (VT, VN, P, S)$ – КС-грамматика.

Нетерминальный символ $A \in VN$ называется непроизводящим, если множество терминальных цепочек, выводимых из него, пусто, обозначается $\{\alpha \mid \alpha \in VT^*, A \Rightarrow* \alpha\} = \emptyset$.

Символ, принадлежащий словарю грамматики V, называется недостижимым, если он не может быть получен ни в одной из сентенциальных форм. Непроизводящие и недостижимые символы называются *бесполезными*. Грамматика называется *приведенной*, если из неё удалены все правила, содержащие бесполезные символы.

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: $N \Rightarrow+ \mu, \mu \in VT^+$.

Если символ в левой части правила достижим, то и символы правой части правила достижимы. Иначе говоря, для любого нетерминала $A \in VN$ должно быть справедливо: $S \Rightarrow* \alpha A \beta$, где $\alpha, \beta \in V^*$.

1.1.1 Удаление непроизводящих символов

Неформально процесс удаления непроизводящих символов может быть представлен следующим образом. 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. Алгоритм удаления непроизводящих символов представлен ниже.

Пусть задана исходная КС-грамматика $G = (VT, VN, P, S)$. Необходимо построить эквивалентную ей грамматику $G' = (VT, VN', P', S)$ такую, что:

- VN' содержит только нетерминальные символы, из которых могут быть выведены цепочки VT', т.е. для всех нетерминалов $A \in VT'$ существует цепочка $\alpha \in VT^*$ такая, что $A \Rightarrow^* \alpha$; и
- P' содержит только те правила, все символы которых являются элементами множества $VT \cup VN'$.

Множество VN' может быть получено посредством следующего алгоритма 4.3. Затем в множество правил P' помещаются только те правила из исходного множества P, у которых все нетерминальные символы в левой и правой частях принадлежат множеству VN'. Обычно, если $A \in VN'$, и $A \to \varepsilon \in P$, то правило $A \to \varepsilon$ помещается в P'.

Алгоритм 4.3 Удаление непроизводящих символов
Вход $G = (VT, VN, P, S)$
Выход $G' = (VT, VN', P', S)$

```
VN' = Ø
flag =1
while(flag){
        flag = 0
        foreach A → α in P {
            if(α ∈ (VT ∪ VN')*)
                {VN'.add(A); flag:=1}}
    }
```

Пример 4.6. Рассмотрим грамматику $G = (\{a, b\}, \{A, B, C, D, E, F, S\}, P, S)$.

Исходное множество правил P	Формирование множества VN'	Результирующая грамматика
$S \to AC \mid BS \mid B$	$VN'_0 = \{\emptyset\}$	$G' = (VT, VN', P', S)$
$A \to aA \mid aF$	$VN'_1 = \{B, F\}$	$VT = \{a, b\}$
$B \to CF \mid b$	$VN'_2 = \{B, F, A, S\}$	$VN' = \{A, B, E, F, S\}$
$C \to cC \mid D$	$VN'_3 = \{B, F, A, S, E\}$	$P' = \{S \to BS \mid B$
$D \to aD \mid BD \mid C$	$VN'_4 = \{B, F, A, S, E\}$	$A \to aA \mid aF$
$E \to aA \mid BSA$	$VN'_4 = VN'_3$	$B \to b$
$F \to bB \mid b$	FINISH	$E \to aA \mid BSA$
		$F \to bB \mid b\}$

1.1.2 Удаление недостижимых символов

Неформально процесс удаления из словаря V исходной грамматики недостижимых символов может быть описан следующим образом. Начальный символ грамматики S считается достижимым. Далее последовательно просматриваются правила, в левой части которых находится достижимый нетерминальный символ, и все символы в 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.

Пусть задана исходная КС грамматика G = (VT, VN, P, S). Необходимо построить эквивалентную ей грамматику G’ = (VT’, VN’, P’, S) такую, что символ из (VT’ ∪ VN’) может быть получен в сентенциальной форме грамматики, иначе говоря для, всех символов $x \in (VT' \cup VN')$ существуют $\alpha, \beta \in (VT' \cup VN')^*$ такие, что $S \Rightarrow^* \alpha x \beta$.

Множества VT’ и VN’ могут быть получены как результат работы следующего алгоритма.

```
Алгоритм 4.4 Удаление недостижимых символов
Вход G = (VT, VN, P, S).
Выход G’ = (VT, VN’, P’, S)
      VN’ = {S}
      VT’ = ∅
      flag = 1
      while(flag){
           foreach A → α in P {
              flag := 0
              foreach x in α {
                  if (A ∈ VN’ && x ∈ VN)
                     {VN’.add(x) flag = 1}
                  if (A ∈ VN’ && x ∈ VT)
                      VT’.add(x)
      }}}
```

Затем множество правил P’ конструируется только из тех правил исходного множества P, все символы которого принадлежат (VT’ ∪ VN’). Обычно, если $A \in VN'$, и $A \to \varepsilon \in P$, то правило $A \to \varepsilon$ помещается в P’.

Исходная грамматика из примера 4.6, к которой были последовательно применены алгоритмы 4.3 и 4.4, становится грамматикой без бесполезных символов, т. е. *приведенной*. Нетрудно убедится, что полученная приведенная грамматика из примера 4.7 порождает язык $L(G') = \{b^n \mid n > 0\}$.

Важным является последовательность применения алгоритмов 4.3 и 4.4. Применение этих же алгоритмов в обратной последовательности не гарантирует удаление из грамматики бесполезных символов, а значит, грамматику нельзя считать приведенной.

Пример 4.7 В качестве исходной берем результирующую грамматику из примера 4.6.

Исходное множество правил	Формирование множеств VT' и VN'	Результирующая грамматика
$S \to BS \mid B$	$VN'_0 = \{S\}$ $VT'_0 = \emptyset$	
$A \to aA \mid aF$	$VN'_1 = \{S, B\}$ $VT'_1 = \{b\}$	$G' = (VT', VN', P', S)$
$B \to b$	$VN'_2 = \{S, B\}$ $VT'_2 = \{b\}$	$VT' = \{b\}$
$E \to aA$ I BSA	$VN'_2 = VN'_1$	$VN' = \{A, B, E, F, S\}$
$F \to bB \mid b$	FINISH	$P' = \{S \to BS \mid B$
		$B \to b\}$

Пусть задана КС-грамматика $G = (VT, VN, P, S)$; если $A \to B \in P$ – правило грамматики, а символы A и $B \in VN$, то такое правило называется *цепным*. Дальнейшее упрощение грамматик может быть осуществлено в процессе удаления ε-правил и цепных правил. Важно понимать, что под упрощением понимается не уменьшение количества правил грамматики или символов её словаря, а повышение однотипности этих правил, что упрощает проектирование распознавателей, при этом количество правил грамматики, а также нетерминалов, скорее всего, увеличится.

1.1.3 Удаление ε-правил

Контекстно-свободная грамматика $G = (VT, VN, P, S)$ называется расширенной КС грамматикой, если в множестве правил P содержится одно или более ε-правила (правила вида $A \to \varepsilon$).

Грамматика называется S-расширенной КС грамматикой (VT, VN, P, S), если множество P содержит правило $S \to \varepsilon$.

Для любой расширенной КС грамматики G, такой что $\varepsilon \notin L(G)$, существует эквивалентная КС грамматика G' без ε-правил.

Для любой расширенной КС грамматики G, такой что $\varepsilon \in L(G)$, существует эквивалентная S-расширенная грамматика G'.

Наличие ε-правил не является критичным для описания грамматики, но может усложнять процесс построения распознавателя для языка, определяемого этой грамматикой. В общем случае наличие ε-правил не является необходимым, хотя они могут быть получены в результате формальных преобразований. Удаление ε-правил для грамматики G позволяет упрощать процесс построения распознавателя для языка L(G), однако множество P правил грамматики может существенно усложниться.

Для заданной КС грамматики G нетерминальный символ A называется nullable, если $A \Rightarrow^* \varepsilon$, иначе говоря, из A выводима пустая цепочка. Множество Nullable для грамматики G включает в себя все нетерминалы, из которых выводимы пустые цепочки.

Алгоритм 4.5 Построение множества Nullable для нетерминалов грамматики

Вход грамматика $G = (VT, VN, P, S)$ и $A \in VN$

Выход множество Nullable(A)

```
Nullable = ∅
flag = 1
while(flag){
        flag = 0
        foreach A → α in P {
            if (α = ε)
                {Nullable.add(A); flag = 1}

            if (α ∈ VN⁺ && forall Bi in α : Bi ∈ Nullable )
                {Nullable.add(A); flag = 1}
    }}
```

После того, как было построено множество Nullable для нетерминальных символов грамматики, можно приступать к устранению ε-правил.

Алгоритм 4.6 Удаление ε-правил

Вход расширенная КС грамматика G = (VT, VN, P, S)

Выход КС грамматику G' = (VT, VN, P', S) без ε-правил

Шаг 1. Построение множества Nullable см. Алгоритм 4.5; P' = ∅

Шаг 2. P' = P - {(A→ε) ∈ P для всех A ∈ VN }.

Шаг 3. Если S ∈ Nullable поместить S → ε в P'

Шаг 4. Для всех оставшихся правил из P' вида заменить каждое правило $A \to x_1 \ldots x_n$ для любых n>0 всеми возможными правилами следующего вида: $A \to y_1 \ldots y_n$, где:

$y_i = x_i$ or $y_i = \varepsilon$ для всех x_i в правой части правила $\{x_1 \ldots x_n\}$ которые принадлежат Nullable

$y_i = x_i$ для всех x_i в правой части правила $\{x_1 \ldots x_n\}$ которые не принадлежат Nullable

Пусть задана исходная расширенная КС грамматика G = (VT, VN, P, S), требуется построить эквивалентную ей S-расширенную КС грамматику G' = (VT, VN, P', S), в которой правило S → ε присутствует тогда и только тогда ε ∈ L(G).

На первом шаге алгоритма 4.6 строится множество Nullable для нетерминальных символов грамматики. На втором шаге в множество правил P' помещаются все правила, кроме ε-правил. Шаг три выполняется только при условии наличия в P правила S → ε. На четвертом шаге необходимо пополнить множество P' правилами, которые получаются путем удаления из их правой части всех возможных комбинаций нетерминальных символов, принадлежащих множеству Nullable.

Пример 4.8. Рассмотрим в качестве исходной грамматику G = (VN, VT, P, S), где:

VN = {S, A, B, C}

VT = {a, b, c}

P {S → ACA

A → aAa | B | C

B → bB | b

C → cC | ε}

Шаг 1. Построение множество Nullable

Итерация	Множество Nullable
0	∅
1	{C}
2	{A, C}
3	{A, C, S}
4	{A, C, S}

На четвертом шаге не было добавлено ни одного нового символа в Nullale.

Шаг 2.	Шаг 3.
P’ = $S \to ACA$ $A \to aAa \mid B \mid C$ $B \to bB \mid b$ $C \to cC$	Добавляем $S \to \varepsilon$ в P’.

Шаг 4.

Исходный набор правил P’	Результирующий набор правил P’
$S \to \varepsilon \mid ACA$	$S \to \varepsilon \mid ACA \mid AA \mid AC \mid CA \mid A \mid C$
$A \to aAa \mid B \mid C$	$A \to aAa \mid aa \mid B \mid C$
$B \to bB \mid b$	$B \to bB \mid b$
$C \to cC$	$C \to cC \mid c$

1.1.4 Удаление цепных правил.

Цепные правила удлиняют процесс вывода сентенциальных форм, усложняют грамматику и не являются обязательными. Цепное правило $A \to B$ показывает, что любая цепочка, выводимая из B, может быть выведена и из A. Идею удаления цепных правил рассмотрим на следующем примере:

Рассмотрим следующие правила	Цепное правило	Замена правой части правила $A \to B$ на все возможные правые части правил с B в левой части правила
$A \to aA \mid a \mid B$ $B \to bB \mid b \mid C$	$A \to B$	$A \to aA \mid a \mid bB \mid b \mid C$ $B \to bB \mid b \mid C$

Пусть задана грамматика G = (VT, VN, P, S) без ε-правил. Требуется построить грамматику G’ = (VT, VN, P’, S) без цепных правил. Множество P’ содержит все нецепные правила из P, вместе со всеми подстановками вида $A \to \alpha$, если $A \Rightarrow^* B$ посредством только цепных правил, и $B \to \alpha$.

Алгоритм 4.7 Построение множества Chain для нетерминала A из неукорачивающейся КС G = (VT, VN, P, S)
Вход КС G = (VT, VN, P, S) и нетерминал A
Выход Множество Chain(A)

```
Chain(A) = {A}
Prev = ∅
while(Chine(A) != Prev){
        Temp = Chain(A) - Prev
        Prev = Chain(A)
        foreach B in Temp
              foreach B → C in P
                    Chain(A).add(C)
}
```

Вывод $A \Rightarrow* C$, содержащий только цепные правила, назовем цепным. Для каждого нетерминала из грамматики G найдем множество всех нетерминальных символов, для которых существует цепной вывод. Обозначим это множество Chain(A), где $A \in VN$. Алгоритм 4.7 строит множество Chain для нетерминального символа.

Вспомогательное множество Temp содержит нетерминальные символы, которые были добавлены в множество Chain на предыдущем шаге работы алгоритма.

Множество нетерминальных символов, принадлежащих Chain(A), определяет замены, которые необходимо сделать для удаления цепных правил, в левой части которых находится нетерминальный символ A.

Следующий алгоритм позволяет получить грамматику G' без цепных правил, эквивалентную грамматике G.

Алгоритм 4.8 Построение КС грамматики без цепных правил
Вход: неукорачивающейся (без ε-правил) КС G = (VT, VN, P, S)
Выход: G' = (VT, VN, P', S) без цепных правил

```
P' = P - {(A → α) ∈ P | ∀ α ∉ VN}       //все кроме цепных правил
foreach A in VN{
     if (Chain(A) != {A}){
          foreach B in Chain(A)
               foreach B → α  in P
                    if (α ∉ VN )
                         P'.add(A → α)}
}
```

Пример 4.9.

Рассмотрим в качестве исходной грамматику G = ({a, b, c}, {A, B, C, S}, P, S)

P = { $S \to ACA \mid AA \mid AC \mid CA \mid A \mid C$

$A \to aAa \mid aa \mid B \mid C$

$B \to bB \mid b$

$C \to cC \mid c$ }

Действие	Множество P’
$P' = P - \{(A \to \alpha) \in P$ для всех $\alpha \notin VN\}$	$S \to ACA \mid AA \mid AC \mid CA$ $A \to aAa \mid aa$ $B \to bB \mid b$ $C \to cC \mid c$
Chain(S) = {S, A, C, B} Добавляем правила с $S \to aAa \mid aa$ $S \to bB \mid b$ $S \to cC \mid c$	$S \to ACA \mid AA \mid AC \mid CA \mid aAa \mid aa \mid bB \mid b \mid cC \mid$ $A \to aAa \mid aa$ $B \to bB \mid b$ $C \to cC \mid c$
Chain(A) = {A, C, B} Добавляем правила с $A \to bB \mid b$ $A \to cC \mid c$	$S \to ACA \mid AA \mid AC \mid CA \mid aAa \mid aa \mid bB \mid b \mid cC \mid$ $A \to aAa \mid aa \mid bB \mid b \mid cC \mid c$ $B \to bB \mid b$ $C \to cC \mid c$
Chain(B) = {B} Chain(C) = {C}	Без изменений Без изменений

В результате множество правил результирующей грамматики имеет следующий вид:

$S \to ACA \mid AA \mid AC \mid CA \mid aAa \mid aa \mid bB \mid b \mid cC \mid c$

$A \to aAa \mid aa \mid bB \mid b \mid cC \mid c$

$B \to bB \mid b$

$C \to cC \mid c$

Удаление цепных правил приводит к общему увеличению числа правил грамматики, но упрощает вывод её сентенциальных форм.