Алгоритмы для контекстно-свободных грамматик

Некоторые обозначения, используемые при описании алгоритмов $A \leftarrow B$ — операция присваивания («A присвоить B»); $A \hookleftarrow a$ — операция добавления элемента в множество («добавить в множество A элемент a»); $A \hookleftarrow B$ — операция добавления множества элементов во множество («добавить в множество A все элементы из B»); $\triangleright$ — начало комментария, продолжающегося до конца строки.

Алгоритм 1 (нахождение порождающих символов)**.**
Вход: КС-грамматика $G = (\Sigma, N, \mathcal{P}, S \in N)$.
Выход: $\mathrm{Gen}_G(\Sigma)$ — множество порождающих в G символов.

1 $\mathrm{Gen}_G(\Sigma) \leftarrow \Sigma$ $\triangleright$ Терминалы являются порождающими символами
  $\triangleright$ Ищем продукции, в правых частях которых только порождающие символы...
2 **while** $\exists A \to X_1 \dots X_n \in \mathcal{P}$, $n \geqslant 0$, такое что $\forall i\ X_i \in \mathrm{Gen}_G(\Sigma)$
    do $\mathrm{Gen}_G(\Sigma) \hookleftarrow A$ $\triangleright$ ...левые части таких продукций добавляются в $\mathrm{Gen}_G(\Sigma)$:

Замечание (о многократном просмотре продукций)**.** В цикле на шаге 2 одна и та же продукция $A \to X_1 \dots X_n$ может быть просмотрена несколько раз, причём если на ранних итерациях она не удовлетворяла условию $\forall i\ X_i \in \mathrm{Gen}_G(\Sigma)$, то на поздних, когда множество $\mathrm{Gen}_G(\Sigma)$ станет достаточно большим, положение дел может измениться (условие $\forall i\ X_i \in \mathrm{Gen}_G(\Sigma)$ выполнится и A надо будет добавить в $\mathrm{Gen}_G(\Sigma)$).

Алгоритм 2 (нахождение достижимых символов)**.**
Вход: КС-грамматика $G = (\Sigma, N, \mathcal{P}, S \in N)$.
Выход: Reach_G — множество достижимых в G символов.

1 $\mathrm{Reach}_G \leftarrow \{S\}$ $\triangleright$ S достижим в G
  $\triangleright$ Ищем продукции, в левых частях которых стоит достижимый символ...
2 **while** $\exists A \to X_1 \dots X_n \in \mathcal{P}$, такое что $A \in \mathrm{Reach}_G$
  $\triangleright$ ...символы из правых частей таких продукций добавляются в Reach_G:
    do $\mathrm{Reach}_G \hookleftarrow \{X_i\}_{i=1}^{n}$

Замечание (о многократном просмотре продукций)**.** Справедливо замечание, аналогичное сделанному в алгоритме 1

Алгоритм 3 (удаление бесполезных символов)**.**
Вход: КС-грамматика $G = (\Sigma, N, \mathcal{P}, S \in N)$.
Выход: КС-грамматика $G' = (\Sigma', N', \mathcal{P}', S \in N')$, не содержащая бесполезных символов, такая что $L(G') = L(G)$, либо сигнал о том, что язык исходной грамматики пуст и не существует эквивалентной G грамматики без бесполезных символов.

1 Построить множество $\mathrm{Gen}_G(\Sigma)$, используя алгоритм 1. Если $S \notin \mathrm{Gen}_G(\Sigma)$ — завершение алгоритма, сообщение о пустоте языка исходной грамматики. Иначе удалить из G все символы, не вошедшие в $\mathrm{Gen}_G(\Sigma)$.

2 Построить множество Reach_G, используя алгоритм 2. Удалить из G все символы, не вошедшие в Reach_G. Получившуюся грамматику обозначить G' и подать её на выход алгоритма.

Замечание (об операции удаления символа из грамматики)**.** Когда в алгоритме требуется удалить символ X из грамматики G, необходимо не только исключить его из множества N или Σ, но и *удалить из $\mathcal{P}$ все продукции, в которых он участвует.*

Алгоритм 4 (удаление ε-правил)**.**
Вход: КС-грамматика $G = (\Sigma, N, \mathcal{P}, S \in N)$.
Выход: КС-грамматика $G' = (\Sigma, N, \mathcal{P}', S \in N)$, без ε-правил (продукций вида $A \to \varepsilon$), такая что $L(G') = L(G) \setminus \{\varepsilon\}$.
Метод: «устранение перегородок».

1 Построить множество $\mathrm{Gen}_G(\varepsilon) \subset N$ всех порождающих ε нетерминалов, используя следующую процедуру:

1 **for** $A \to \varepsilon \in \mathcal{P}$
  do $\mathrm{Gen}_G(\varepsilon) \hookleftarrow A$
2 **while** $\exists A \to X_1 \dots X_n \in \mathcal{P}$, где $\{X_i\}_{i=0}^n \subset \mathrm{Gen}_G(\varepsilon)$
  do $\mathrm{Gen}_G(\varepsilon) \hookleftarrow A$

2 Выполнить следующие действия:

for $A \to \alpha_0 B_1 \alpha_1 \dots B_n \alpha_n \in \mathcal{P}$, где $\forall i\ B_i \in \mathrm{Gen}_G(\varepsilon) \wedge\ \alpha_i \in ((N \cup \Sigma) \setminus \mathrm{Gen}_G(\varepsilon))^*$
  do $\mathcal{P} \hookleftarrow \{\alpha_0 X_1 \alpha_1 \dots X_n \alpha_n \mid \forall i\ X_i = \varepsilon \vee X_i = B_i\}$

3 Удалить из $\mathcal{P}$ все ε-правила, обозначить получившееся множество правил $\mathcal{P}'$ и подать на выход алгоритма грамматику $G' = (N, \Sigma, \mathcal{P}', S)$.

Замечание (о методе «устранения перегородок»)**.** На шаге 2 каждая продукция $\mathcal{P}$ просматривается ровно один раз. Понятно, что для подходящего n любая продукция может быть представлена в виде $A \to \alpha_0 B_1 \alpha_1 \dots B_n \alpha_n$ с указанными условиями для B_i и α_i. Например, если в продукции нет ε-порождающих символов, то это продукция вида $A \to \alpha_0$ и для неё нет необходимости добавлять новые продукции.
Другой пример: продукция $C \to aDbD$, где $D \in \mathrm{Gen}_G(\varepsilon)$, $a, b \in \Sigma$, это продукция вида $A \to \alpha_0 B_1 \alpha_1 B_2 \alpha_2$, где $A = C$, $\alpha_0 = a$, $B_1 = B_2 = D$, $\alpha_2 = \varepsilon$, и для неё нужно добавить три новых продукции. Укажем их, пояснив название метода «удаления перегородок». Можно считать, что ε-порождающие символы B_i являются «перегородками» в продукции $A \to \alpha_0 B_1 \alpha_1 \dots B_n \alpha_n$ и новые продукции получаются из данной при помощи устранения этих перегородок всеми возможными способами. Для продукции $C \to aDbD$ это: $C \to abD$, $A \to aDb$ и $A \to ab$.

Алгоритм 5 (удаление цепных продукций)**.**
Вход: КС-грамматика $G = (\Sigma, N, \mathcal{P}, S \in N)$, не содержащая ε-правил.
Выход: КС-грамматика $G' = (\Sigma, N, \mathcal{P}', S \in N)$, без цепных правил (продукций вида $A \to B$), такая что $L(G') = L(G)$.

1 Для каждого нетерминала $A \in N$ построить множество циклически достижимых из A нетерминалов $C(A)$, используя процедуру:

   $C(A) \leftarrow \{A\}$
   while $\exists D \to E \in \mathcal{P}$, такая что $D \in C(A)$
      do $C(A) \hookleftarrow E$

2 Выполнить следующую процедуру:

   for $A \in N$
      do for $B \to \alpha \in \mathcal{P}$
         do if $B \in C(A)$
            then $\mathcal{P} \hookleftarrow A \to \alpha$

3 Удалить из $\mathcal{P}$ все цепные правила, обозначить получившееся множество правил $\mathcal{P}'$ и подать на выход алгоритма грамматику $G' = (N, \Sigma, \mathcal{P}', S)$.

Алгоритм 6 (приведение к нормальной форме Хомского)**.**
Вход: КС-грамматика $G = (\Sigma, N, \mathcal{P}, S \in N)$, не содержащая ε-правил.
Выход: КС-грамматика $G' = (\Sigma', N', \mathcal{P}', S \in N)$ в НФХ, такая что $L(G') = L(G) \setminus \{\varepsilon\}$ или сообщение о пустоте языка грамматики.
Метод: «разбиение слов на слоги».

1 Последовательно воспользоваться алгоритмами 4 и 5, полученную грамматику обозначить $G'' = (N, \Sigma, \mathcal{P}'', S)$.

2 Применить к G'' алгоритм 3. Если получен ответ $L(G'') = \varnothing$, остановить алгоритм и подать на выход сигнал о пустоте языка исходной грамматики. Иначе получена грамматика $G' = (N', \Sigma', \mathcal{P}', S)$.

3 К G' применить следующую процедуру:

   for $a \in \Sigma$
      do if $\exists A \to X_1 \dots X_n \in \mathcal{P}''$, такая что $n > 1 \wedge \exists i\; X_i = a$
         then
            добавить в N *новый* нетерминал A'
            заменить a на A' во всех продукциях с правой частью длиннее 1
            $\mathcal{P}'' \hookleftarrow A' \to a$

4 К G' применить следующую процедуру:

> **for** $A \to B_1 B_2 \dots B_n \in \mathcal{P}'$, где $n > 2$, $B_i \in N$
>   **do**
>   добавить в N *новые* нетерминалы $C_1, \dots C_{n-2}$
> $\mathcal{P}' \hookleftarrow \{A \to B_1 C_1\} \cup \{C_i \to B_{i+1} C_{i+1}\}_{i=1}^{n-3} \cup \{C_{n-2} \to B_{n-1} B_n\}$
>   удалить $A \to B_1 B_2 \dots B_n$ из $\mathcal{P}'$

Подать G' на выход алгоритма.

Замечание (о введении новых нетерминалов)**.** Следует отметить, что на каждой итерации цикла шага 4, если есть необходимость ввести новые нетерминалы, то они должны отличаться не только от тех, которые присутствовали в грамматике до начала этого цикла, но и от тех, которые были введены на предыдущих итерациях этого цикла. Таким образом, одного комплекта букв $\{C_i\}$ для выполнения цикла может не хватить. Для борьбы с нехваткой букв можно вводить нетерминалы, помеченные частями исходного слова $B_1 \dots B_n$, которое «разбивается на слоги». Например, вместо набора $\{C_i\}_{i=1}^{n-2}$ можно использовать набор $\{\langle B_{i+1} \dots B_n \rangle\}_{i=1}^{n-2}$, где для каждого i выражение $\langle B_{i+1} \dots B_n \rangle$ понимается как *один новый* нетерминал. Аналогично можно поступать на шаге 3, добавляя для рассматриваемого терминала a новый нетерминал $\langle a \rangle$, а не A'.

Алгоритм 7 (решение проблемы принадлежности для КС-языков; Кок—Янгер—Касами, CYK-алгоритм)**.**
Вход: грамматика $G = (\Sigma, N, \mathcal{P}, S \in N)$ в НФХ, слово $w = w_1 \dots w_n \in \Sigma^*$.
Выход: да, $w \in L(G)$ / нет, $w \notin L(G)$.
Метод: последовательное определение нетерминалов, выводящих всевозможные подстроки w всё большей длины.
Для всех $1 \leqslant i \leqslant j \leqslant n$ определим множество

$$N_{ij} = \{A \in N \mid A \Rightarrow_G^* w_i \dots w_j\}.$$

Построить множества N_{ij}, используя процедуру:

> **for** $i \leftarrow 1$ **to** n
>   **do** $N_{ii} \leftarrow \{A \in N \mid A \to w_i \in \mathcal{P}\}$ $\triangleright$ Подстроки w длины 1
> **for** $s \leftarrow 2$ **to** n $\triangleright$ Цикл по длине подстроки
>   **do for** $i \leftarrow 1$ **to** $n - s + 1$ $\triangleright$ Цикл по месту начала подстроки
> $j \leftarrow i + s - 1$ $\triangleright$ Позиция конца подстроки с началом в w_i длины s
> $N_{ij} \leftarrow \{A \in N \mid A \to BC \in \mathcal{P};\ \exists k \in [i, j-1]_{\mathbb{Z}} \colon B \in N_{ik},\ C \in N_{k+1j}\}$

Если $S \in N_{1n}$, то подать на выход алгоритма «да», иначе — «нет».

Замечание (о табличной форме CYK-алгоритма)**.** Алгоритм удобно выполнять, заполняя таблицу с N_{ij} в ячейках. Ячейки таблицы расположены в системе координат (i, s), в позиции (i, s) находится множество $N_{i,i+s-1}$.