

Формальные грамматики. HW#2

Тураев Тимур, SPbAU, SE, 604 group

1. Построить обыкновенную грамматику в нормальном виде Хомского для языка Дика $D = \{\varepsilon, ab, aabb, abab, aaabbb, \dots\}$ над алфавитом $\{a, b\}$. Для этой грамматики и для входной строки $w = abaabba \notin D$, построить таблицу разбора $T_{i,j}$, как в алгоритме Кокка–Касами–Янгера.

Обыкновенная грамматика не в нормальной форме:

$$S \rightarrow aSb \mid SS \mid \varepsilon$$

Приведем эту грамматику к нормальной форме Хомского:

- Удалим длинное правило $S \rightarrow aSb$

$$\begin{aligned} S &\rightarrow Tb \mid SS \mid \varepsilon \\ T &\rightarrow aS \end{aligned}$$

- Удалим ε -правила:

$$\begin{aligned} S &\rightarrow C \mid \varepsilon \\ C &\rightarrow Tb \mid CC \\ T &\rightarrow a \mid aC \end{aligned}$$

- Удалим цепные правила:

$$\begin{aligned} S &\rightarrow Tb \mid CC \mid \varepsilon \\ C &\rightarrow Tb \mid CC \\ T &\rightarrow a \mid aC \end{aligned}$$

- Последний шаг: заменим терминалы на нетерминалы (бесполезных символов в грамматике нет):

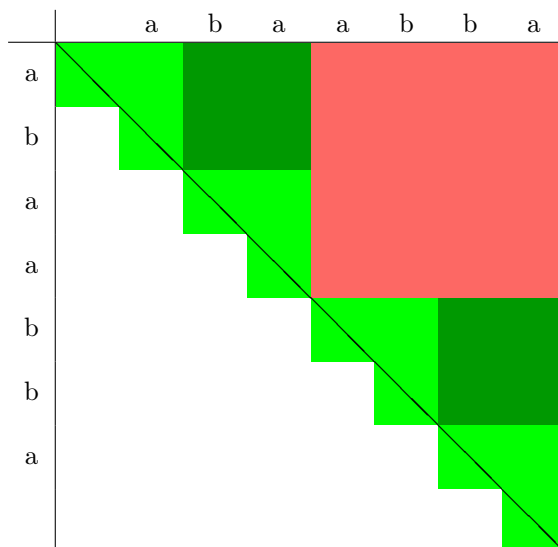
$$\begin{aligned} S &\rightarrow TB \mid CC \mid \varepsilon \\ C &\rightarrow TB \mid CC \\ T &\rightarrow a \mid AC \\ A &\rightarrow a \\ B &\rightarrow b \end{aligned}$$

Таблица разбора в алгоритме Кокка–Касами–Янгера:

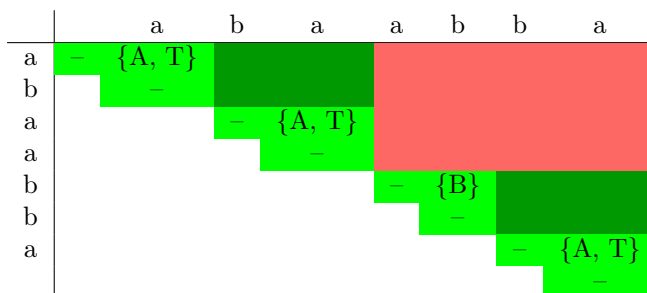
	a	b	a	a	b	b	a
a	{A, T}	{S, C}	\emptyset	\emptyset	\emptyset	{S, C}	\emptyset
b		{B}	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset
a			{A, T}	\emptyset	{T}	{S, C}	\emptyset
a				{A, T}	{S, C}	\emptyset	\emptyset
b					{B}	\emptyset	\emptyset
b						{B}	\emptyset
a							{A, T}

По значению отсутствию нетерминала S в $T_{0,7}$ видно что да, данная строка не принадлежит языку.

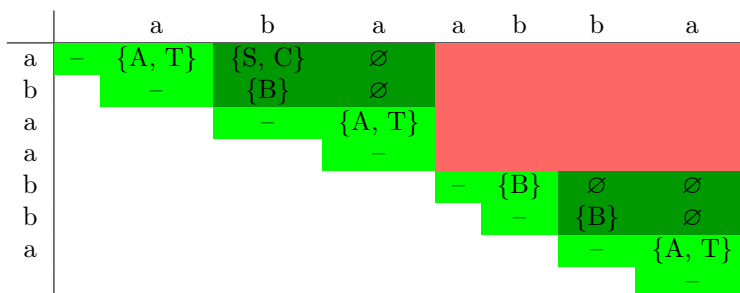
2. Рассмотреть работу алгоритма Валианта для грамматики, построенной в прошлом упражнении. Среди всех действий, производимых алгоритмом, найти то произведение булевых матриц, после вычисления которого станет верным условие $S \in f(P_{0,6})$, где S — начальный символ грамматики. Описать, когда и как именно вычисляется это произведение — то есть, какая процедура, вызванная с какими значениями, и какой оператор в ней умножает какие две булевы матрицы какого размера, каков результат умножения, и какие элементы $P_{i,j}$ будут этим затронуты?



Рассмотрим работу алгоритма Валианта. Единственной функцией, вызванной из $main()$, будет $compute(0,8)$. Она вызовет $compute(0,4)$ и $compute(4,8)$ на двух больших зеленых треугольниках. Эти две функции, в свою очередь, запустят $compute(0,2)$, $compute(2,4)$, $compute(4,6)$, $compute(6,8)$, который обещают посчитать маленькие светло-зеленые треугольники; и успешно их обчисляют: каждая такая функция $compute(k, k+2)$ запустит единственную $complete(k, k+1, k+1, k+2)$, которая, выполнив 19-я строку алгоритма, заполнит все $T_{i,j}$, стоящие на светло-зеленых клетках. Получится следующее:



Кроме обсчета маленьких светло-зеленых треугольников, функции $compute(0,4)$ и $compute(4,8)$ обчисляют и темно-зеленые квадраты:



Таким образом, функции $compute(0,4)$ и $compute(4,8)$ (внутри вызова $compute(0,8)$) закончат свою работу. Далее вызовется функция **complete(0, 4, 4, 8)**. Она поделит большой красный квадрат на 4 квадрата D, E, C, D' и запустит $complete(C)$, то есть $complete(2, 4, 4, 6)$. Эта функция его обчисляет аналогично тому как обсчитывались темно-зеленые квадраты. 9-я строка алгоритма внутри **complete(0, 4, 4, 8)** выполнена.

	a	b	a	a	b	b	a
a	-	{A, T}	{S, C}	∅	D		E
b		-	{B}	∅			
a			-	{A, T}	∅	{T}	D'
a				-	{A, T}	{S, C}	
b					-	{B}	∅
b						-	{B}
a							-

Далее выполняются строки 10-11 (обсчет матрицы D) и строки 12-13 (обсчет матрицы D') внутри **complete(0, 4, 4, 8)**.

	a	b	a	a	b	b	a
a	-	{A, T}	{S, C}	∅	∅	∅	E
b		-	{B}	∅	∅	∅	
a			-	{A, T}	∅	{T}	{S, C}
a				-	{A, T}	{S, C}	∅
b					-	{B}	∅
b						-	{B}
a							-

Наконец алгоритм перейдет к строке **14**: $P_E = P_E \cup (T_B \times T_{D'})$. Подматрица P_E это матрица $\begin{pmatrix} P_{0,6} & P_{0,7} \\ P_{1,6} & P_{1,7} \end{pmatrix}$. Подматрица T_B на рисунке выше – это первая темно-зеленая, то есть $\begin{pmatrix} P_{0,2} & P_{0,3} \\ P_{1,2} & P_{1,3} \end{pmatrix} = \begin{pmatrix} \{S, C\} & \emptyset \\ \{B\} & \emptyset \end{pmatrix}$, а матрица T'_D это темно-красная матрица $\begin{pmatrix} P_{2,6} & P_{2,7} \\ P_{3,6} & P_{3,7} \end{pmatrix} = \begin{pmatrix} \{S, C\} & \emptyset \\ \emptyset & \emptyset \end{pmatrix}$. Именно их алгоритм собирается перемножать, используя произведение булевых матриц.

Алгоритм работает так: по каждой паре нетерминалов $\{A, B\}$ (их у нас $|N|^2 = 25$) строятся две булевы матрицы: матрица X^A , элемент которой (булев бит) означает есть ли нетерминал A в соответствующем элементе первой матрицы; и матрица Y^B , элемент которой (булев бит) означает есть ли нетерминал B в соответствующем элементе второй матрицы. Получившаяся в результате умножения булева матрица, которую можно назвать Z^{AB} , означает «где в итоговой матрице $(T_B \times T_{D'})$ будет элемент-пара нетерминалов $\{A, B\}$ ».

В нашем случае, когда дело дойдет до рассмотрения пары $\{C, C\}$, матрица X^C будет иметь вид $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$, а матрица Y^C $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$. Их произведение: $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$, что означает, пара $\{C, C\}$ гарантированно будет стоять на первой строке первого столбца матрицы P_E . А это в свою очередь означает, что когда алгоритм дойдет до 16 строки ($complete(E)$), затем вызовется ($complete(D) = complete(0, 1, 6, 7)$), в матрице T на месте $(0, 6)$ появится нетерминал S , потому что существует правило $S \rightarrow CC$ (вместе с ним, в этом же множестве нетерминалов, из-за пары $\{C, C\}$ появится еще и C , так как существует правило $C \rightarrow CC$).

Итак, собираем все вместе и отвечаем на вопросы в задании:

- какая процедура с какими параметрами – **complete(0, 4, 4, 8)** (вызванная из $compute(0, 8)$, которая в свою очередь вызвана из $main()$)
- какой оператор в ней – 14-я строка алгоритма, $(T_B \times T_{D'})$
- какие две булевы матрицы какого размера – умножаются 2 матрицы 2×2 : $X^C = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$ и $Y^C = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$
- каков результат умножения этих булевых матриц – $Z^{CC} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$
- общий результат умножения $(T_B \times T_{D'}) - \begin{pmatrix} \{S, S\}, \{C, C\}, \{S, C\}, \{C, S\} \\ \{B, S\}, \{B, C\} \end{pmatrix} \emptyset$
- какие элементы $P_{i,j}$ будут этим затронуты – при этом умножении затронуты будут элементы подматрицы $P_E = \begin{pmatrix} P_{0,6} & P_{0,7} \\ P_{1,6} & P_{1,7} \end{pmatrix}$

3. Замкнут ли класс LL языков относительно пересечения с регулярными языками? Если замкнут, привести построение, а если незамкнут, привести пример LL грамматики и регулярного языка с доказательством несуществования LL грамматики для их пересечения

Рассмотрим язык L_1 , который задает все строки четной длины, первая половина которых состоит только из букв a , а вторая – из любых сочетаний букв b и c :

$$L_1 = \{a^n(b|c)^n \mid n \geq 0\}$$

Это $LL(1)$ -язык, для него можно построить такую грамматику:

$$\begin{aligned} S &\rightarrow aSB \mid \varepsilon \\ B &\rightarrow b \mid c \end{aligned}$$

и следующую таблицу разбора:

	a	b	c	ε
S	$S \rightarrow aSB$	$S \rightarrow \varepsilon$	$S \rightarrow \varepsilon$	$S \rightarrow \varepsilon$
B	–	$B \rightarrow b$	$B \rightarrow c$	$B \rightarrow \varepsilon$

Наряду с L_1 рассмотрим язык L_2 , задающий все слова, в котором сначала идет какое-то (возможно нулевое) число букв a , а затем какое-то (возможно нулевое) число букв b :

$$L_2 = \{a^n b^m \mid n, m \geq 0\}$$

Это тоже $LL(1)$ -язык, для него можно построить такую грамматику:

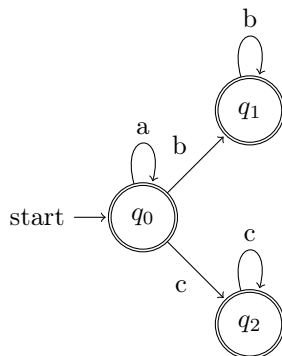
$$\begin{aligned} S &\rightarrow AE \\ A &\rightarrow aA \mid \varepsilon \\ E &\rightarrow bB \mid cC \mid \varepsilon \\ B &\rightarrow bB \mid \varepsilon \\ C &\rightarrow cC \mid \varepsilon \end{aligned}$$

и следующую таблицу разбора:

	a	b	c	ε
S	$S \rightarrow AE$	$S \rightarrow AE$	$S \rightarrow AE$	$S \rightarrow AE$
E	–	$E \rightarrow bB$	$E \rightarrow cC$	$E \rightarrow \varepsilon$
B	–	$B \rightarrow bB$	–	$B \rightarrow \varepsilon$
C	–	–	$C \rightarrow cC$	$C \rightarrow \varepsilon$

Кроме того, этот язык, очевидно, является регулярным:

$$L_2 = a^*(b^* \mid c^*)$$



Пересечением этих языков является язык L_3 :

$$L_1 \cap L_2 = L_3 = \{a^n b^n + a^n c^n \mid n \geq 0\}$$

который, как мы знаем, не является $LL(k)$ ни для какого k (см пример 8.4 конспекта 11 лекции). Отсюда делаем вывод, что класс LL языков не замкнут относительно пересечения с регулярными языками.

4. Построить линейную грамматику для языка $f(L_0)$, где $L_0 = \{w\$w^R \mid w \in \{a, b\}^*\}$ и $f(L) = \{[w_{1,1}\# \dots \# w_{1,k_1}] \dots [w_{m,1}\# \dots \# w_{m,i_m} : w_{1,i_1} w_{2,i_2} \dots w_{m,i_m} \in L]\}$ (это $NLOGSPACE$ -полный язык, приведённый на лекции).

5. Разрешима ли такая задача: «по данной обыкновенной грамматике, определить, порождает ли она хотя бы одну строку чётной длины»? Если разрешима, привести алгоритм, а если неразрешима, доказать это с помощью методов лекции 15 (использовав язык $VALC$ в готовом виде, или же определив новый его вариант).

Разрешима.

Приведем данную нам обыкновенную грамматику $G = (\Sigma, N, R, S)$ к нормальному виду Хомского. Далее, предположим, что грамматика не порождает пустое слово (нет правила $S \rightarrow \varepsilon$, иначе все очевидно).

Заведем для каждого $A \in N$ по 2 булевых флага: первый флаг означает, порождает ли A хотя бы одно слово нечетной длины, а второй флаг означает, порождает ли A хотя бы одно слово четной длины. Изначально все флаги для всех нетерминальных символов сброшены.

Рассмотрим правила вида $A \rightarrow \alpha$, где $\alpha \in \Sigma$ и установим флаг «порождает слово нечетной длины» всем таким A .

Далее в алгоритме будет несколько итераций. На каждой итерации для каждого правила $A \rightarrow BC$, в котором для A еще не выставлен флаг «порождает слово нечетной длины», попытаемся его установить следующим образом: если B и C порождают слова длиной разной четности (установлены разные флаги), то флаг выставляем, иначе нет. Абсолютно аналогично действуем для флага «порождает слово четной длины» (там правило будет такое: у B и C установлены оба каких-либо флагов).

Продолжаем итерации до тех пор, пока устанавливается хотя бы один флаг. Алгоритм, очевидно, закончится, ибо флагов и правил конечное число. Кроме того, так как грамматика в нормальной форме, то у нас не существует ни бесполезных символов. Также достаточно ясно, что алгоритм даст верный ответ, это видно из правил установки флагов.