Формальные грамматики. HW#2

Тураев Тимур, SPbAU, SE, 604 group

1. *Постоить 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:

$$S \to aSb \mid SS \mid \varepsilon$$

Приведем эту грамматику к нормальной форме Хомского:

- Удалим длинное правило $S \to aSb$

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

- Удалим ε-правила:

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

- Удалим цепные правила:

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

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

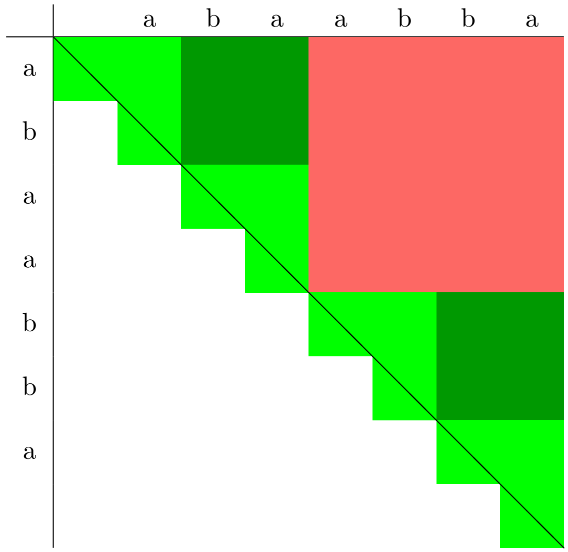
$$\begin{aligned} S &\to TB \mid CC \mid \varepsilon \\ C &\to TB \mid CC \\ T &\to a \mid AC \\ A &\to a \\ B &\to b \end{aligned}$$

Таблица разбора в алгоритме Кокка–Касами–Янгера:

	a	b	a	a	b	b	a
a	{A, T}	{S, C}	∅	∅	∅	{S, C}	∅
b		{B}	∅	∅	∅	∅	∅
a			{A, T}	∅	{T}	{S, C}	∅
a				{A, T}	{S, C}	∅	∅
b					{B}	∅	∅
b						{B}	∅
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 обсчитают: 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:

		a	b	a	a	b	b	a
a	–	{A, T}						
b		–						
a			–	{A, T}				
a				–				
b					–	{B}		
b						–		
a							–	{A, T}
								–

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

		a	b	a	a	b	b	a
a	–	{A, T}	{S, C}	∅				
b		–	{B}	∅				
a			–	{A, T}				
a				–				
b					–	{B}	∅	∅
b						–	{B}	∅
a							–	{A, T}
								–

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 обсчитает аналогично тому как обсчитывались темно-зеленые квадраты. 9-я строка алгоритма внутри **complete(0, 4, 4, 8)** выполнилась.

		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							–	{A, T}
								–

Далее выполнятся строки 10-11 (обсчет матрицы D) и стркои 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							–	{A, T}
								–

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

Алгоритм работает так: по каждой паре нетерминалов $\{A, B\}$ (их у нас $|N|^2 = 25$) строются 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 $\{\mathbf{C}, \mathbf{C}\}$, матрица X^C будет иметь вид $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$, а матрица Y^C $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$. Их произведение: $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$, 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 $S \to CC$ (вместе с ним, в этом же множестве нетерминалов, из-за пары $\{C, C\}$ появится еще и C, так как существует правило $C \to CC$).

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

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

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:

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

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

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

и следующую таблицу разбора:

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

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:

$$L_2 = \{a^n b^m + a^n c^m \mid n, m \geqslant 0\}$$

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

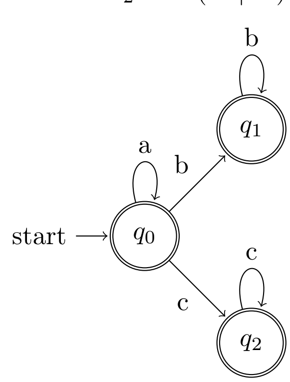
$$\begin{aligned} S &\to AE \\ A &\to aA \mid \varepsilon \\ E &\to bB \mid cC \mid \varepsilon \\ B &\to bB \mid \varepsilon \\ C &\to cC \mid \varepsilon \end{aligned}$$

и следующую таблицу разбора:

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

Кроме того, этот язык, очевидно, является регулярным:

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



Пересечением этих языков является язык L_3:

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

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* $L_0 = \{w\$w^R \mid w \in \{a,b\}^*\}$
$f(L) = \{[w_{1,1}\# \dots \# w_{1,k_1}] \dots [w_{m,1}\# \dots \# w_{m,k_m}] \mid \exists i_1, \dots, i_m :\ w_{1,i_1} w_{2,i_2} \dots w_{m,i_m} \in L\}$
S это правильная строка, то есть строка, заключенная в квадратные скобки.

$$S \to [A]$$

Внутри это строки делаем что угодно, при условии, что в каждом блоке, заключенный в квадратные скобки, будет возможность выбрать пустую строку, то есть никак не испортить итоговое слово $w_{1,i_1} w_{2,i_2} \dots w_{m,i_m}$:

$$\begin{aligned}
&A \to][A \mid A][\mid B \mid \#C \mid B' \mid C'\# \mid L\\
&B \to aB \mid bB \mid \#B \mid \#\#C \mid D\\
&C \to aC \mid bC \mid \#C \mid][A\\
&D \to aD \mid bD \mid \#D \mid \#][A\\
&B' \to B'a \mid B'b \mid B'\# \mid C'\#\# \mid D'\\
&C' \to C'a \mid C'b \mid C'\# \mid A][\\
&D' \to D'a \mid D'b \mid D'\# \mid A][\#
\end{aligned}$$

Наконец, когда-нибудь дойдем до правила $A \to L$. Правило L' (как и R') дописывает к уже имеющейся последовательности с кусочком полупалиндрома (либо к пустой его части) любую последовательность букв и решеток:

$$\begin{aligned}
&L \to L' \mid F\\
&L' \to aL' \mid bL' \mid \#L' \mid \#R\\
&R \to R' \mid F\\
&R' \to R'a \mid R'b \mid R'\# \mid F\#
\end{aligned}$$

Наконец, правило F порождает либо \$, либо буквы a или b слева от знака доллара и запускает генерацию бесполезных блоков с самого начала. Кроме того, правило O генерирует бесполезные подблоки уже после вставленных букв a или b.

$$\begin{aligned}
&F \to \$ \mid aFa \mid bFb \mid][A][\mid][C'\# \mid \#C][\mid \#O\#\\
&O \to aO \mid bO \mid \#O \mid Oa \mid Ob \mid O\# \mid][A][
\end{aligned}$$

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 $S \to \varepsilon$, 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 $A \to \alpha$, 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 $A \to BC$, в котором для A еще не выставлен флаг «порождает слово нечетной длины», попытаемся его уставновить 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 бесполезных символов. Также достаточно ясно, что алгоритм даст верный ответ, это видно из правил установки флагов.

6. *Разрешима ли такая задача: «по данной обыкновенной грамматике, определить, порождает ли она хотя бы одну строку-палиндром w, т.е., строку, для которой $w = w^R$»?*

Не разрешима.

Для данной нам Тьюринг-машины построим грамматики $G_1 = (\Sigma, N_1, R_1, S_1)$ и $G_2 = (\Sigma, N_2, R_2, S_2)$ так, как это сделано в лемме 11.1.

Построим грамматику $G_2^R = (\Sigma, N_2^R, R_2^R, S_2^R)$, такую, что $L(G_2^R) = \{w \mid w^R \in L(G_2)\}$, то есть грамматику, порождающую все слова в языке $L(G_2)$, но записанные в обратном порядке. Это сделать довольно просто: так как грамматика G_2 линейна (и даже LL(1)), то достаточно просто записать все ее правила в обратном порядке. Более того, так как большая часть правил симметрична, то переписать придется лишь 2 правила – первое $S_2 \to aS_2$ перепишется в $S_2 \to S_2a$ и правило $E \to \$E\#$ перепишется в $E \to \#E\$$ (все остальные правила останутся без изменений, см. конец доказательства леммы 11.1). Далее, переобозначим все нетерминальные символы, дописав у каждого букву ^{R}.

Далее, построим новую грамматику $G = (\Sigma \cup \{\%\}, N_1 \cup N_2^R \cup \{S\}, R_1 \cup R_2^R \cup R, S)$ $(\% \notin \Sigma)$, которая комбинирует грамматики G_1 и G_2^R в соответствии со следующим правилом:

$$S \to S_1\%S_2^R$$

Нетрудно понять, какой язык порождает эта грамматика: $L(G) = \{u\%v \mid u \in L(G_2) \wedge v \in L(G_2^R)\}$, то есть все слова, записанные через знак процента, левая часть которых (до знака процента) – слово из языка, порождаемого первой грамматикой, а развернутая правая часть – слово из языка, порождаемого второй грамматикой.

Грамматика G порождает хотя бы один палиндром тогда и только тогда, когда найдется хотя бы одно слово, которое принадлежит и $L(G_1)$ и $L(G_2)$. Иными словами поставленная задача разрешима тогда и только тогда, когда разрешима задача определения $L(G_1) \cap L(G_2) = \varnothing$?. А эта задача неразрешима по теореме 11.1. Значит и исходная задача тоже неразрешима.