В работе [1] предложен алгоритм синтаксического анализа, основанный на перемножении матриц. Для того, чтобы описать модификацию этого алгоритма, в разделе 1 мы введем немного измененную терминологию, дальше, в разделе 2 переформулируем оригинальный алгоритм в рамках этой терминологии. Обещанную модификацию алгоритма опишем в секции 3.

1 Терминология

Введем обозначения, немного отличающиеся от используемых в статье [1]. Пусть $G = (\Sigma, N, R, S)$ — грамматика в нормальной форме Хомского и $w = a_1 \dots a_n$ — строка, причем $n + 1 = 2^k$. Ограничение на длину строки (для обоих рассмотренных алгоритмов) введены скорее из соображений простоты изложения, так как оба алгоритма довольно легко обобщаются на строки произвольной длины. Далее, пусть T — треугольная матрица $n \times n$ с элементами $T[i, j] \subseteq N;\ 1 \leqslant i, j \leqslant n$. Цель алгоритма — заполнить ячейки матрицы T так, чтобы выполнялось следующее условие:

$$T[i, j] = \{A \,|\, a_{n+1-j} \dots a_i \in L(A)\}, \quad \text{при } 1 \leqslant i, j \leqslant n < i + j\,.$$

Вспомогательная матрица P с элементами $P[i, j] \subseteq N \times N$ в результате должна быть заполнена значениями

$$P[i, j] = \{(B, C) \,|\, a_{n+1-j} \dots a_i \in L(B)L(C)\}, \quad 1 \leqslant i, j \leqslant n < i + j\,.$$

Определение 1.1. Будем называть (i, j) *корректной* парой индексов, если для них выполнены условия $1 \leqslant i, j \leqslant n < i + j$.

Заметим, что алгоритм в принципе рассматривает только те ячейка матриц T и P, которые задаются корректными парами индексов.

Определение 1.2. Назовем *(квадратной) подматрицей* такой набор корректных пар индексов $S = \{(i, j)\}$, что существуют корректная пара индексов (a, b) и $size > 0$, для которых выполнены следующие условия: $size \leqslant n + 1 - a$, $size \leqslant n + 1 - b$, а также любая пара (i, j), удовлетворяющая условиям $a \leqslant i < a + size$ и $b \leqslant j < b + size$, принадлежит множеству S. Тогда $size$ — *размер*, а пара индексов (a, b) — *вершина* этой подматрицы.

Определение 1.3. Для подматрицы m с вершиной (a, b) и размером s определим множество индексов $\mathcal{I}_m$, *влияющее на подматрицу* m, как

$$\mathcal{I}_m = \{(i,j) \,|\, i < n+1-b,\, j < b+s\} \cup \{(i,j) \,|\, i < a+s,\, j < n+1-a\}\,.$$

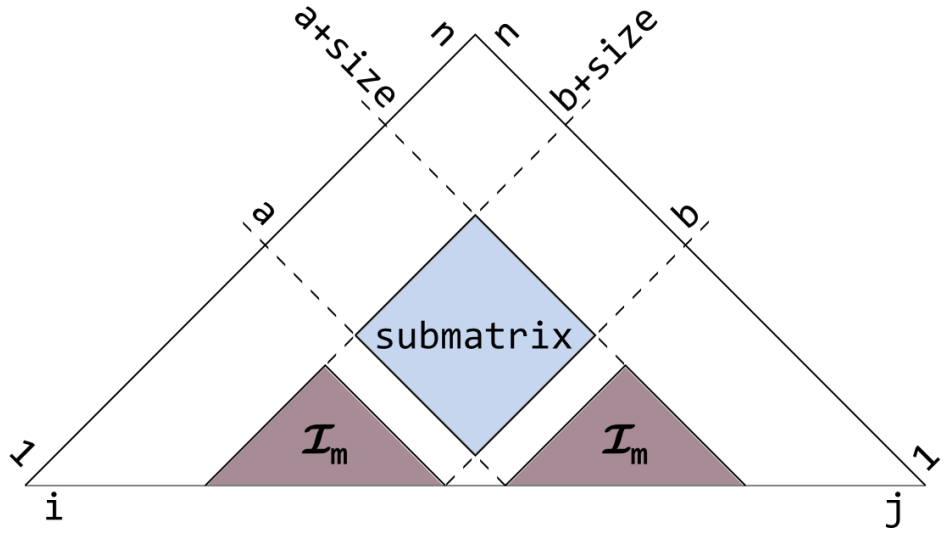

Рис. 1: Иллюстрация к определениям 1.2 и 1.3: (a, b) — вершина подматрицы submatrix размером (size×size); $\mathcal{I}_m$ — зона, влияющая на подматрицу submatrix

Определение 1.4. Пусть S — подматрица. Будем обозначать через $T[S]$ и $P[S]$ подматрицы матриц T и P, соответствующие множеству индексов S.

Для описания алгоритмов нам необходимо задать ряд вспомогательных функций для оперирования подматрицами. Эти функции представлены в листинге 1.

Листинг 1: Вспомогательные функции обработки подматриц

```
1  function size(m)
2  function bottomCell(m)
3
4  function leftSubmatrix(m)
5  function topSubmatrix(m)
6  function rightSubmatrix(m)
7  function bottomSubmatrix(m)
8
9  function rightNeighbor(m)
10 function leftNeighbor(m)
11 function rightGrounded(m)
12 function leftGrounded(m)
```

Функции $size(m)$ и $bottomCell(m)$ вычисляют размер и вершину подматрицы m соответственно. Следующие четыре функции $leftSubmatrix(m)$ — $bottomSubmatrix(m)$ возвращают одну из подматриц, делящих исходную подматрицу m на четыре части, как показано на рисунке 2.

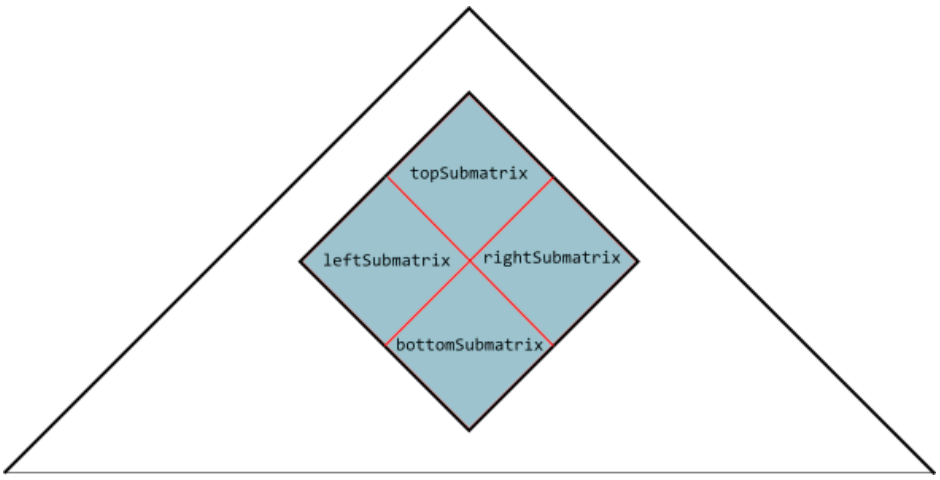

Рис. 2: Иллюстрация к листингу 1: разбиение матрицы на 4 компоненты

В свою очередь функции $rightNeighbor(m)$ и $leftNeighbor(m)$ возвращают подматрицы, сдвинутые относительно исходной на $size(m)$ по одной из осей, как показано на рисунке 3. Функции $rightGrounded(m)$ и $leftGrounded(m)$ (рисунок 3) также возвращают подматрицы, сдвинутые относительно m, но на этот раз величина сдвига вычисляется по формуле $a + b - (n + 1)$, где (a, b) — вершина матрицы m. Заметим, что для вершины (i, j) любой их двух матриц $rightGrounded(m)$ и $leftGrounded(m)$ выполняется равенство $i + j = n + 1$. Это означает, что вершина расположена в самой «нижней» из использующихся диагоналей всей матрицы.

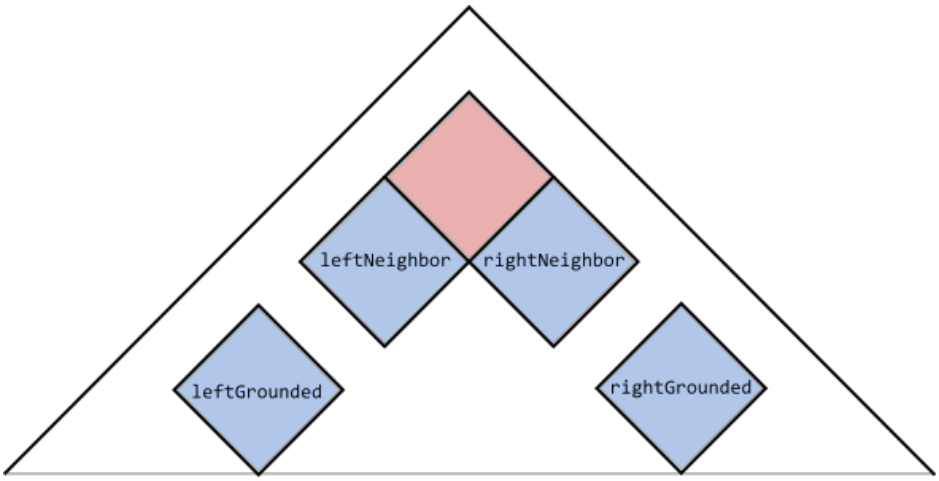

Рис. 3: Иллюстрация к листингу 1: соседи матрицы

2 Алгоритм синтаксического анализа, основанный на перемножении матриц

Описание алгоритма, взятого из статьи [1], представлено в виде псевдокода, приведённого в следующем листинге 2.

Здесь, процедура $compute(i, j)$ принимает на вход такие i, j, что $(i - 1, j - 1)$ — корректная пара индексов и записывает значения во все ячейки (i', j') матрицы T такие, что $i' + j' > n$, $i' < i$ и $j' < j$.

Процедура $complete(m)$, в свою очередь, принимает на вход подматрицу m и определена только для подматриц с размером, являющимся

степенью двойки. $complete(m)$ вычисляет все значения матрицы T на переданном ей множестве индексов m, при выполнении некоторых дополнительных условий. Во-первых, матрица T должны быть корректно заполнена для всех пар индексов из множества $\mathcal{I}_m$. Во-вторых для всех $(i,j) \in m$ текущее значение $P[i,j]$ должно быть следующим:

$$\{(B,C) \mid \exists k : n+1-b \leqslant k < a,\ a_{n+1-j} \dots a_k \in L(B) \text{ и } a_{k+1} \dots a_i \in L(C)\},$$

где (a,b) — вершина подматрицы m.

Листинг 2: Алгоритм синтаксического анализа, основанный на перемножении матриц

```
1  procedure main():
2      compute(n+1, n+1)
3
4  procedure compute(i, j):
5      if i + j − n ⩾ 4 then
6          compute (i, j/2)
7          compute (i/2, j)
8      denote m = submatrixByBottomCellAndSize ((n+i−j)/2, (n+j−i)/2), (i+j−n)/2)
9      complete(m)
10
11 procedure complete(m)
12     denote (i, j) = bottomCell(m)
13     if size(m) = 1 and i + j = n then
14         T[i, j] = {A | A → a_i ∈ R}
15     else if size(m) = 1 then
16         T[i, j] = f(P[i, j])
17     else if size(m) > 1 then
18         denote B = bottomSubmatrix(m), L = leftSubmatrix(m),
19                R = rightSubmatrix(m), T = topSubmatrix(m)
20         complete(B)
21         P[L] = P[L] ∪ (T[leftGrounded(L)] × T[B])
22         complete(L)
23         P[R] = P[R] ∪ (T[B] × T[rightGrounded(R)])
24         complete(R)
25         P[T] = P[T] ∪ (T[leftGrounded(T)] × T[R])
26         P[T] = P[T] ∪ (T[L] × T[rightGrounded(T)])
27         complete(T)
```

3 Модифицированный алгоритм

Далее предложена модификация исходного алгоритма синтаксического анализа, предназначенная для его более естественной адаптации к задаче поиска подстрок, выводимых в заданной грамматике.

Главная процедура *main* сначала обрабатывает нижний слой матрицы T (клетки (i, j), для которых $i + j = n + 1$), записывая в него корректные значения. Потом разбивает матрицу T на слои, как показано на картинке 4 (каждый слой состоит из набора подматриц, у каждой из которых отброшена нижняя четверть — $bottomSubmatrix$). Полученные слои обрабатываются последовательно снизу вверх, с помощью процедуры $completeVLayer$, заполняя тем самым всю матрицу T.

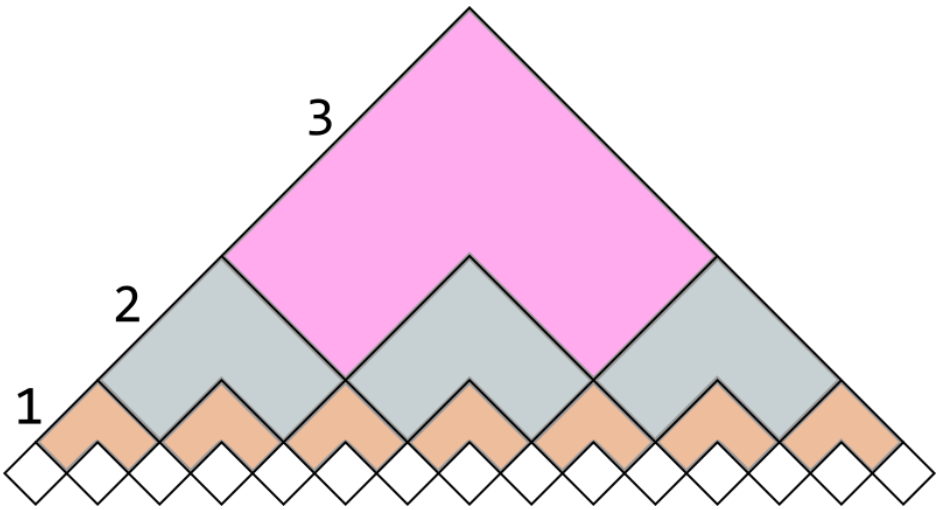

Рис. 4: Первичное разбиение на слои

Процедура $completeVLayer$, в свою очередь, принимает на вход набор подматриц M. Эти подматрицы не должны пересекаться, а также для любых двух подматриц $m_1, m_2 \in M$; (a_i, b_i), s_i — вершина и размер m_i соответственно, должно выполняться $s_1 = s_2$ и $a_1 + b_1 = a_2 + b_2$. Для каждого элемента m множества M процедура достраивает матрицу T для трех верхних четвертей ($leftSubmatrix(m)$, $rightSubmatrix(m)$ и $topSubmatrix(m)$). Для корректной работы этой функции, во-первых, необходимо, чтобы для любой $m \in M$ ячейки $T[i, j]$ были построены для $(i, j) \in bottomSubmatrix(m)$ и для $(i, j) \in \mathcal{I}_m$. Во-вторых, требуется

удовлетворение ограничения на матрицу P, аналогичного ограничению в случае процедуры *complete* оригинального алгоритма, а именно: для любой $m \in M$ и для всех $(i, j) \in m$ текущее значение $P[i, j]$ должно быть следующим:

$$\{(B, C) \mid \exists k : n + 1 - b \leqslant k < a,\ a_{n+1-j} \dots a_k \in L(B) \text{ и } a_{k+1} \dots a_i \in L(C)\},$$

где (a, b) — вершина подматрицы m.

Третья процедура — *completeLayer* — тоже принимает на вход набор подматриц M, но для каждого элемента m этого набора достраивает матрицу T для всей подматрицы m. Ограничения на входные данные такие же, как и у процедуры *completeVLayer*. Для корректной работы этой функции необходимо, чтобы для любой $m \in M$ ячейки $T[i, j]$ были построены для $\mathcal{I}_m$, а так же выполнение того же требования на P, что и в предыдущем случае.

Листинг 3: Алгоритм

```
1  procedure main():
2    for ℓ in (1, ..., n) do
3      T[ℓ, n + 1 − ℓ] = {A | A → a_ℓ ∈ R}
4    foreach 1 ⩽ i < k do
5      denote layer = constructLayer(i)
6      completeVLayer(layer)
7
8  procedure completeLayer(M):
9    if ∀ m ∈ M, size(m) = 1 then
10     denote cells = {bottomCell(m) | m ∈ M}
11     foreach (x, y) ∈ cells do
12       T[x, y] = f(P[x, y])
13   else
14     denote bottomLayer = {bottomSubmatrix(m) | m ∈ M}
15     completeLayer(bottomLayer)
16     completeVLayer(M)
17
18 procedure completeVLayer(M):
19   denote leftSubLayer = {leftSubmatrix(m) | m ∈ M}
20   denote rightSubLayer = {rightSubmatrix(m) | m ∈ M}
21   denote topSubLayer = {topSubmatrix(m) | m ∈ M}
22
23   foreach m ∈ leftSubLayer do
```

```
24      P[m] = P[m] ∪ (T[leftGrounded(m)] × T[rightNeighbor(m)])
25    foreach m ∈ rightSubLayer do
26      P[m] = P[m] ∪ (T[leftNeighbor(m)] × T[rightGrounded(m)])
27
28    completeLayer(leftSubLayer ∪ rightSubLayer)
29
30    foreach m ∈ topSubLayer do
31      P[m] = P[m] ∪ (T[leftGrounded(m)] × T[rightNeighbor(m)])
32    foreach m ∈ topSubLayer do
33      P[m] = P[m] ∪ (T[leftNeighbor(m)] × T[rightGrounded(m)])
34
35    completeLayer(topSubLayer)
```

Процедура *main* реализуется через *completeVLayer* очевидным образом. Для построения множества подматриц, составляющий слой под номером i используется функция $constructLayer(i)$, реализация которой оставлена за скобками. Процедура *completeLayer* по сути аналогична процедуре *complete* из статьи, за исключением того, что она выполняется сразу для нескольких матриц. Таким же образом отдельно разбирается случай, когда переданные матрицы имеют размер 1×1. Иначе матрицы разбиваются на четыре части и сначала следует рекурсивный вызов от $bottomSubmatrix$ (для всех переданных матриц), а затем вызов процедуры *completeVLayer*, которая обрабатывает верхние части матриц.

Процедура *completeVLayer* для каждой из переданных матриц сначала выполняет два перемножения (соответствует 21 и 23 строкам в алгоритме из статьи), затем вызывает *completeLayer* от $rightSubmatrix$ и $leftSubmatrix$ (22 и 24 строки оригинального алгоритма), далее выполняет оставшиеся два умножения (строки 25 и 26) и, наконец, вызывает *completeLayer* от оставшейся части $topSubmatrix$ (строка 27).

4 Заключение

Заметим, что если мы хотим адаптировать алгоритм для задачи поиска подстроки длины не большей, чем какое-то заданное $m \ll n$, тогда нам надо строить матрицы T и P только для индексов $i + j \leqslant n + m$. В этом случае в оригинальном алгоритме придется производить большое количество «холостых» рекурсивных вызовов. В модифицированном

алгоритме этого отчасти можно избежать, изменив верхнюю границу в цикле for (строка 4 листинга 3).

Также представленная модификация алгоритма собирает вместе все умножения матриц одного размера (и расположенных на одном «уровне»), которые можно сделать независимо друг от друга. Это позволяет более эффективно распараллеливать эти умножения.

Список литературы

[1] Alexander Okhotin, “Parsing by matrix multiplication generalized to Boolean grammars”, *Theoretical Computer Science*, V. 516, p. 101–120, January 2014