

# Приложения теории формальных языков и синтаксического анализа

Семён Григорьев

6 сентября 2019 г.

# Содержание

<b>1</b>	<b>Введение</b>	<b>4</b>
<b>2</b>	<b>Общие сведения теории графов</b>	<b>4</b>
2.1	Транзитивное замыкание графа . . . . .	4
2.2	Задачи поиска путей . . . . .	4
2.3	Вопросы и задачи . . . . .	5
<b>3</b>	<b>Общие сведения теории формальных языков</b>	<b>5</b>
3.1	Контекстно-свободные грамматики и языки . . . . .	6
3.2	Дерево вывода . . . . .	8
3.3	Пустота КС-языка . . . . .	9
3.4	Нормальная форма Хомского . . . . .	9
3.5	Вопросы и задачи . . . . .	11
<b>4</b>	<b>Постановка задачи о поиске путей с ограничениями в терминах формальных языков</b>	<b>11</b>
4.1	Вопросы и задачи . . . . .	11
<b>5</b>	<b>О разрешимости задачи поиска путей с ограничениями в терминах языков</b>	<b>12</b>
5.1	Вопросы и задачи . . . . .	12
<b>6</b>	<b>Алгоритм на матричных операциях</b>	<b>12</b>
6.1	Алгоритм СУК . . . . .	12
6.2	Алгоритм для графов на основе СУК . . . . .	12
6.3	Алгоритм на основе матриц . . . . .	12
6.4	Конъюнктивные и булевы грамматики . . . . .	12
6.4.1	Определения . . . . .	12
6.4.2	Для графов . . . . .	13
6.5	Особенности реализации матричного алгоритма . . . . .	13
6.6	Обзор . . . . .	13
6.7	Вопросы и задачи . . . . .	13
<b>7</b>	<b>Через тензорное произведение</b>	<b>13</b>
7.1	Рекурсивные автоматы и сети . . . . .	14
7.2	Тензорное произведение . . . . .	14
7.3	Алгоритм . . . . .	17
7.4	Примеры . . . . .	17
7.5	Замечания о реализации . . . . .	38
7.6	Вопросы и задачи . . . . .	38

<b>8</b>	<b>Сжатое представление леса разбора</b>	<b>38</b>
8.1	Лес разбора как представление контекстно-свободной грамматики . . . . .	38
8.2	Вопросы и задачи . . . . .	38
<b>9</b>	<b>Алгоритм на основе восходящего анализа</b>	<b>39</b>
9.1	Восходящий синтаксический анализ . . . . .	39
9.2	Вопросы и задачи . . . . .	39
<b>10</b>	<b>Алгоритм на основе нисходящего анализа</b>	<b>39</b>
10.1	Нисходящий синтаксический анализ . . . . .	40
10.2	GLL для КС запросов . . . . .	40
10.3	Вопросы и задачи . . . . .	40
<b>11</b>	<b>Комбинаторы для КС апросов</b>	<b>40</b>
11.1	Парсер комбинаторы . . . . .	40
11.2	Комбинаторы для КС . . . . .	40
11.3	Вопросы и задачи . . . . .	40
<b>12</b>	<b>От CFPQ к вычислению Datalog-запросов</b>	<b>40</b>
12.1	Datalog . . . . .	40
12.2	КС-запрос как запрос на Datalog . . . . .	40
12.3	Обобщение GLL для вычисления Datalog-запросов . . . . .	40

# 1 Введение

Одна из классических задач, связанных с анализом графов — это поиск путей в графе. Возможны различные формулировки этой задачи. В некоторых случаях необходимо выяснить, существует ли путь с определёнными свойствами между двумя выбранными вершинами. В других же ситуациях необходимо найти все пути в графе, удовлетворяющие некоторым свойствам.

Так или иначе, на практике часто требуется указать, что интересующие нас пути должны обладать каким-либо специальными свойствами. Иными словами, наложить на пути некоторые ограничения. Например, указать, что искомый путь должен быть простым, кратчайшим, гамильтоновым и так далее.

Один из способов задавать ограничения на пути в графе основан на использовании формальных языков. Базовое определение языка говорит нам, что язык — это множество слов над некоторым алфавитом. Если рассмотреть граф, рёбра которого помечены символами из алфавита, то путь в графе будет задавать слово: достаточно соединить последовательно символы, лежащие на рёбрах пути. Множество же таких путей будет задавать множество слов или язык. Таким образом, если мы хотим найти некоторое множество путей в графе, то в качестве ограничения можно описать язык, который должно задавать это множество. Иными словами, задача поиска путей может быть сформулирована следующим образом: необходимо найти такие пути в графе, что слова, получаемые конкатенацией меток их рёбер, принадлежат заданному языку. Такой класс задач будем называть задачами поиска путей с ограничениями в терминах формальных языков.

Рассмотрим различные варианты постановки задачи.

Различные алгоритмы решения.

## 2 Общие сведения теории графов

основные определения: рёбра, вершины, графы ориентированные, неориентированные, помеченные.

Специальные понятия, необходимые для изложения конкретного материала, будут даны в соответствующих главах.

Матрицы смежности

### 2.1 Транзитивное замыкание графа

Флойд-Уоршал, матрицы.

Рассуждения про субкубичность. Про то, что булево полукольцо.

### 2.2 Задачи поиска путей

Достижимость, все пути, один путь.

Про кратчайшие, простые и прочее.

Про классические задачи и классическое сведение к матричным операциям.

Тра-та-та.

## 2.3 Вопросы и задачи

1. Реализуйте алгоритм построения транзитивного замыкания через матрицы. Реализовать матрицы самим. Взять готовую библиотеку матричных операций: CPU, GPGPU.
2. Реализуйте поиск кратчайших путей через матричные операции. Реализовать перемножение матриц самостоятельно. Взять готовую библиотеку матричных операций: CPU, GPGPU.

## 3 Общие сведения теории формальных языков

В данной главе мы рассмотрим основные понятия из теории формальных языков, которые пригодятся нам в дальнейшем изложении.

**Определение 3.1.** *Алфавит* — это конечное множество. Элементы этого множества будем называть *символами*.

**Пример 3.1.** Примеры алфавитов

- Латинский алфавит  $\Sigma = \{a, b, c, \dots, z\}$
- Кириллический алфавит  $\Sigma = \{a, б, в, \dots, я\}$
- Алфавит чисел в шестнадцатеричной записи  $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

Традиционное обозначение для алфавита —  $\Sigma$ . Также мы будем использовать различные прописные буквы латинского алфавита. Для обозначения символов алфавита будем использовать строчные буквы латинского алфавита:  $a, b, \dots, x, y, z$ .

Будем считать, что над алфавитом  $\Sigma$  всегда определена операция конкатенации  $(\cdot) : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ . При записи выражений символ точки (обозначение операции конкатенации) часто будем опускать:  $a \cdot b = ab$ .

**Определение 3.2.** *Слово* над алфавитом  $\Sigma$  — это конечная конкатенация символов алфавита  $\Sigma$ :  $\omega = a_0 \cdot a_1 \cdot \dots \cdot a_m$ , где  $\omega$  — слово, а для любого  $i$   $a_i \in \Sigma$ .

**Определение 3.3.** *Слово* над алфавитом  $\Sigma$  — это результат конечной конкатенации символов алфавита  $\Sigma$ :  $\omega = a_0 \cdot a_1 \cdot \dots \cdot a_m$ , где  $\omega$  — слово, а для любого  $i$   $a_i \in \Sigma$ . Будем называть  $m + 1$  длиной слова и обозначать как  $|\omega|$ .

**Определение 3.4.** *Язык* над алфавитом  $\Sigma$  — это множество слов над алфавитом  $\Sigma$ .

**Пример 3.2.** Примеры языков

- Язык целых чисел в двоичной записи  $\{0, 1, -1, 10, 11, -10, -11, \dots\}$
- Язык всех правильных скобочных последовательностей  $\{(), (()), ()(), (())(), \dots\}$

Любой язык над алфавитом  $\Sigma$  является подмножеством  $\Sigma^*$  — множества всех слов над алфавитом  $\Sigma$

Заметим, что язык не обязан быть конечным множеством, в то время как алфавит всегда конечен и изучаем мы конечные слова.

### Определение 3.5. Способы задания языков

- Перечислить для конечных
- Генератор
- Распознаватель

## 3.1 Контекстно-свободные грамматики и языки

Из всего многообразия нас будут интересовать прежде всего контекстно-свободные грамматики.

**Определение 3.6.** Контекстно-свободная грамматика — это четвёрка вида  $\langle \Sigma, N, P, S \rangle$ , где

- $\Sigma$  — это терминальный алфавит;
- $N$  — это нетерминальный алфавит;
- $P$  — это множество правил или продукций, таких что каждая продукция имеет вид  $N_i \rightarrow \alpha$ , где  $N_i \in N$  и  $\alpha \in \{\Sigma \cup N\}^* \cup \varepsilon$ ;
- $S$  — стартовый нетерминал. Отметим, что  $\Sigma \cap N = \emptyset$ .

**Пример 3.3.** Грамматика, задающая язык целых чисел в двоичной записи без лидирующих нулей:  $G = \langle \{0, 1, -\}, \{S, N, A\}, P, S \rangle$ , где  $P$  определено следующим образом:

$$\begin{aligned} S &\rightarrow 0 \mid N \mid -N \\ N &\rightarrow 1A \\ A &\rightarrow 0A \mid 1A \mid \varepsilon \end{aligned}$$

При спецификации грамматики часто опускают множество терминалов и нетерминалов, оставляя только множество правил. При этом нетерминалы часто обозначаются прописными латинскими буквами, терминалы — строчными, а стартовый нетерминал обозначается буквой  $S$ . Мы будем следовать этим обозначениям, если не указано иное.

**Определение 3.7.** Отношение непосредственной выводимости. Мы говорим, что последовательность терминалов и нетерминалов  $\gamma\alpha\delta$  непосредственно выводится из  $\gamma\beta\delta$  при помощи правила  $\alpha \rightarrow \beta$  ( $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$ ), если

- $\alpha \rightarrow \beta \in P$
- $\gamma, \delta \in \{\Sigma \cup N\}^* \cup \varepsilon$

**Определение 3.8.** Отношение выводимости является рефлексивно-транзитивным замыканием отношения непосредственной выводимости

- $\alpha \xRightarrow{*} \beta$  означает  $\exists \gamma_0, \dots, \gamma_k : \alpha \Rightarrow \gamma_0 \Rightarrow \gamma_1 \Rightarrow \dots \Rightarrow \gamma_{k-1} \Rightarrow \gamma_k \Rightarrow \beta$
- Транзитивность:  $\forall \alpha, \beta, \gamma \in \{\Sigma \cup N\}^* \cup \varepsilon : \alpha \xRightarrow{*} \beta, \beta \xRightarrow{*} \gamma \Rightarrow \alpha \xRightarrow{*} \gamma$
- Рефлексивность:  $\forall \alpha \in \{\Sigma \cup N\}^* \cup \varepsilon : \alpha \xRightarrow{*} \alpha$

- $\alpha \xRightarrow{*} \beta$  —  $\alpha$  выводится из  $\beta$
- $\alpha \xRightarrow{k} \beta$  —  $\alpha$  выводится из  $\beta$  за  $k$  шагов
- $\alpha \xRightarrow{+} \beta$  — при выводе использовалось хотя бы одно правило грамматики

**Пример 3.4.** Пример вывода цепочки  $-1101$  в грамматике:

$$\begin{aligned} S &\rightarrow 0 \mid N \mid -N \\ N &\rightarrow 1A \\ A &\rightarrow 0A \mid 1A \mid \varepsilon \end{aligned}$$

$$S \Rightarrow -N \Rightarrow -1A \Rightarrow -11A \xRightarrow{*} -1101A \Rightarrow -1101$$

**Определение 3.9.** *Рефлексивно-транзитивное замыкание отношения* — это наименьшее рефлексивное и транзитивное отношение, содержащее исходное.

**Определение 3.10.** *Вывод слова в грамматике.* Слово  $\omega \in \Sigma^*$  выводимо в грамматике  $\langle \Sigma, N, P, S \rangle$ , если существует некоторый вывод этого слова из начального нетерминала  $S \xRightarrow{*} \omega$ .

**Определение 3.11.** *Левосторонний вывод.* На каждом шаге вывода заменяется самый левый нетерминал.

**Пример 3.5.** Пример левостороннего вывода цепочки в грамматике

$$\begin{aligned} S &\rightarrow AA \mid s \\ A &\rightarrow AA \mid Bb \mid a \\ B &\rightarrow c \mid d \end{aligned}$$

$$S \Rightarrow AA \Rightarrow BbA \Rightarrow cbA \Rightarrow cbAA \Rightarrow cbaA \Rightarrow cbaa$$

Аналогично можно определить правосторонний вывод.

**Определение 3.12.** *Язык, задаваемый грамматикой* — множество строк, выводимых в грамматике  $L(G) = \{\omega \in \Sigma^* \mid S \xRightarrow{*} \omega\}$ .

**Определение 3.13.** Грамматики  $G_1$  и  $G_2$  называются *эквивалентными*, если они задают один и тот же язык:  $L(G_1) = L(G_2)$

**Пример 3.6.** Пример эквивалентных грамматик для языка целых чисел в двоичной системе счисления.

$$\begin{aligned} \Sigma &= \{0, 1, -\} \\ N &= \{S, N, A\} \end{aligned}$$

$$\begin{aligned} S &\rightarrow 0 \mid N \mid -N \\ N &\rightarrow 1A \\ A &\rightarrow 0A \mid 1A \mid \varepsilon \end{aligned}$$

$$\begin{aligned} \Sigma &= \{0, 1, -\} \\ N &= \{S, A\} \end{aligned}$$

$$\begin{aligned} S &\rightarrow 0 \mid 1A \mid -1A \\ A &\rightarrow 0A \mid 1A \mid \varepsilon \end{aligned}$$

**Определение 3.14.** *Неоднозначная грамматика* — грамматика, в которой существует 2 и более выводов для одного слова.

**Пример 3.7.** Неоднозначная грамматика для правильных скобочных последовательностей

$$S \rightarrow (S) \mid SS \mid \varepsilon$$

**Определение 3.15.** *Однозначная грамматика* — грамматика, в которой существует не более одного вывода для каждого слова.

**Пример 3.8.** Однозначная грамматика для правильных скобочных последовательностей

$$S \rightarrow (S)S \mid \varepsilon$$

**Определение 3.16.** *Существенно неоднозначные языки* — язык, для которого невозможно построить однозначную грамматику

**Пример 3.9.** Пример существенно неоднозначного языка

$$\{a^n b^n c^m \mid n, m \in \mathbb{Z}\} \cup \{a^n b^m c^m \mid n, m \in \mathbb{Z}\}$$

## 3.2 Дерево вывода

В некоторых случаях не достаточно знать порядок применения правил. Необходимо структурное представление вывода цепочки в грамматике. Таким представлением является *дерево вывода*.

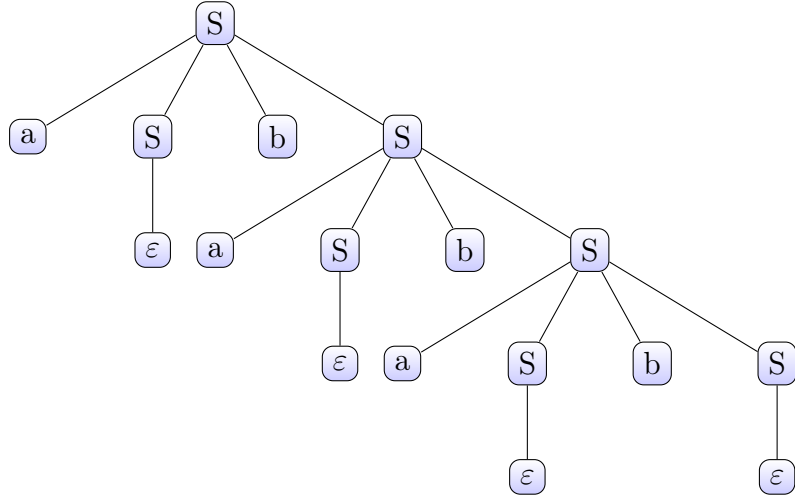
**Определение 3.17.** Деревом вывода цепочки  $\omega$  в грамматике  $G = \langle \Sigma, N, S, P \rangle$  называется дерево, удовлетворяющее следующим свойствам.

1. Помеченное: метка каждого внутреннего узла — нетерминал, метка каждого листа — терминал или  $\varepsilon$ .
2. Корневое: корень помечен стартовым нетерминалом.
3. Упорядоченное.
4. В дереве есть узел с меткой  $N_i$  и сыновьями  $M_j \dots M_k$  тогда и только тогда, когда в грамматике есть правило вида  $N_i \rightarrow M_j \dots M_k$ .
5. Крона образует цепочку.

**Пример 3.10.** Построим дерево вывода цепочки  $ababab$  в грамматике

$$G = \langle \{a, b\}, \{S\}, S, S \rightarrow a S b S, S \rightarrow \varepsilon \rangle$$





**Теорема 3.1.** Пусть  $G = \langle \Sigma, N, P, S \rangle$  — КС-грамматика. Вывод  $S \xRightarrow{*} \alpha$ , где  $\alpha \in (N \cup \Sigma)^*$ ,  $\alpha \neq \varepsilon$  существует  $\Leftrightarrow$  существует дерево вывода в грамматике  $G$  с кроной  $\alpha$ .

### 3.3 Пустота КС-языка

**Теорема 3.2.** Существует алгоритм, определяющий, является ли язык, порождаемый КС грамматикой, пустым

*Доказательство.* Следующая лемма утверждает, что если в КС языке есть выводимое слово, то существует другое выводимое слово с деревом вывода не глубже количества нетерминалов грамматики. Для доказательства теоремы достаточно привести алгоритм, последовательно строящий все деревья глубины не больше количества нетерминалов грамматики, и проверяющий, являются ли такие деревья деревьями вывода. Если в результате работы алгоритма не удалось построить ни одного дерева, то грамматика порождает пустой язык.  $\square$

**Лемма 3.3.** Если в данной грамматике выводится некоторая цепочка, то существует цепочка, дерево вывода которой не содержит ветвей длиннее  $m$ , где  $m$  — количество нетерминалов грамматики

*Доказательство.* Рассмотрим дерево вывода цепочки  $\omega$ . Если в нем есть 2 узла, соответствующих одному нетерминалу  $A$ , обозначим их  $n_1$  и  $n_2$ .

Предположим,  $n_1$  расположен ближе к корню дерева, чем  $n_2$ .

$S \xRightarrow{*} \alpha A_{n_1} \beta \xRightarrow{*} \alpha \omega_1 \beta$ ;  $S \xRightarrow{*} \alpha \gamma A_{n_2} \delta \beta \xRightarrow{*} \alpha \gamma \omega_2 \delta \beta$ , при этом  $\omega_2$  является подцепочкой  $\omega_1$ .

Заменим в изначальном дереве узел  $n_1$  на  $n_2$ . Полученное дерево является деревом вывода  $\alpha \omega_2 \delta$ .

Повторяем процесс замены одинаковых нетерминалов до тех пор, пока в дереве не останутся только уникальные нетерминалы.

В полученном дереве не может быть ветвей длины большей, чем  $m$ .

По построению оно является деревом вывода.  $\square$

### 3.4 Нормальная форма Хомского

**Определение 3.18.** Контекстно-свободная грамматика  $\langle \Sigma, N, P, S \rangle$  находится в *Нормальной Форме Хомского*, если она содержит только правила следующего вида:

- $A \rightarrow BC$ , где  $A, B, C \in N^*$
- $A \rightarrow a$ , где  $A \in N, a \in \Sigma$
- $S \rightarrow \varepsilon$

**Теорема 3.4.** Любую КС грамматику можно преобразовать в НФХ.

*Доказательство.* Алгоритм преобразования в НФХ состоит из следующих шагов:

- Замена неодинокных терминалов
- Удаление длинных правил
- Удаление  $\varepsilon$ -правил
- Удаление цепных правил
- Удаление бесполезных нетерминалов

То, что каждый из этих шагов преобразует грамматику к эквивалентной, при этом является алгоритмом, доказано в следующих леммах.  $\square$

**Лемма 3.5.** Для любой КС-грамматики можно построить эквивалентную, которая не содержит правила с неодинокными терминалами.

*Доказательство.* Каждое правило  $A \rightarrow B_0 B_1 \dots B_k, k \geq 1$  заменить на множество правил:

- $A \rightarrow C_0 C_1 \dots C_k$
- $\{C_i \rightarrow B_i \mid B_i \in \Sigma, C_i \text{ — новый нетерминал}\}$

$\square$

**Лемма 3.6.** Для любой КС-грамматики можно построить эквивалентную, которая не содержит правил длины больше 2.

*Доказательство.* Каждое правило  $A \rightarrow B_0 B_1 \dots B_k, k \geq 2$  заменить на множество правил:

- $A \rightarrow B_0 C_0$
- $C_0 \rightarrow B_1 C_1$
- $\dots$
- $C_{k-3} \rightarrow B_{k-2} C_{k-2}$
- $C_{k-2} \rightarrow B_{k-1} B_k$

$\square$

**Лемма 3.7.** Для любой КС-грамматики можно построить эквивалентную, не содержащую  $\varepsilon$ -правил.

*Доказательство.* Определим  $\varepsilon$ -правила:

- $A \rightarrow \varepsilon$
- $A \rightarrow B_0 \dots B_k, \forall i : B_i \text{ — } \varepsilon\text{-правило.}$

Каждое правило  $A \rightarrow B_0 B_1 \dots B_k$  заменяем на множество правил, где каждое  $\varepsilon$ -правило удалено во всех возможных комбинациях.  $\square$

**Лемма 3.8.** Можно удалить все цепные правила

**Лемма 3.9.** Можно удалить все бесполезные нетерминалы

### 3.5 Вопросы и задачи

1. Предъявить несколько выводов для одной цепочки.
2. Построить выводы
3. Построить деревья вывода !!! Перенести из раздела про SPFF

## 4 Постановка задачи о поиске путей с ограничениями в терминах формальных языков

Пусть  $\mathcal{G} = \langle V, E, L \rangle$  — конечный ориентированный граф с метками на рёбрах, где  $V$  — конечное множество вершин,  $E$  — конечное множество рёбер,  $L$  — конечное множество или алфавит меток. Рёбра будем представлять в виде упорядоченных троек из  $V \times L \times V$ . Путём  $\pi$  в графе  $\mathcal{G}$  будем называть последовательность рёбер такую, что для любых двух последовательных рёбер  $e_1 = (u_1, l_1, v_1)$  и  $e_2 = (u_2, l_2, v_2)$  в этой последовательности, конечная вершина первого ребра является начальной вершиной второго, то есть  $v_1 = u_2$ . Будем обозначать путь из вершины  $v_0$  в вершину  $v_n$  как  $v_0 \pi v_n = e_0, e_1, \dots, e_{n-1} = (v_0, l_0, v_1), (v_1, l_1, v_2), \dots, (v_{n-1}, l_n, v_n)$ .

Функция  $\omega(\pi) = \omega((v_0, l_0, v_1), (v_1, l_1, v_2), \dots, (v_{n-1}, l_n, v_n)) = l_0 \cdot l_1 \cdot \dots \cdot l_n$  строит слово по пути посредством конкатенации меток рёбер вдоль этого пути. Очевидно, для пустого пути данная функция будет возвращать пустое слово, а для пути длины  $n > 0$  — непустое слово длины  $n$ .

Путь  $G = \langle \Sigma, N, P \rangle$  — контекстно-свободная грамматика. Будем считать, что  $L \subseteq \Sigma$ . Мы не фиксируем стартовый нетерминал в определении грамматики, поэтому, чтобы описать язык, задаваемый ей, нам необходимо отдельно зафиксировать стартовый нетерминал. Таким образом, будем говорить, что  $L(G, N_i) = \{w | N_i \xrightarrow{*}_G w\}$  — это язык задаваемый грамматикой  $G$  со стартовым нетерминалом  $N_i$ .

Задача достижимости:

Задача поиска путей:

В задаче поиска путей мы можем накладывать дополнительные ограничения на путь (например, чтобы он был простым или кратчайшим), но это уже другая история.

### 4.1 Вопросы и задачи

1. Пусть есть граф. Задайте грамматику для поиска всех путей, таких, что....
2. Существует ли в графе !!! путь из А в Б, такой что!!!
3. Для графа !!! постройте все пути, удовлетворяющие !!!!

## 5 О разрешимости задачи поиска путей с ограничениями в терминах языков

Сведение к задаче о пересечении с регулярным.

Замкнутость регулярных.

Проверка пустоты.

Замкнутость контекстно-свободных. Проверка пустоты.

Про другие классы языков: конъюнктивные, булевы, многокомпонентные.

### 5.1 Вопросы и задачи

1. Задача 1

2. Задача 2

## 6 Алгоритм на матричных операциях

### 6.1 Алгоритм СҮК

Построение и заполнение таблицы.

### 6.2 Алгоритм для графов на основе СҮК

Данный алгоритм накладывает ограничение на форму грамматики: грамматика должна быть в “ослабленной” нормальной форме Хомского 3.4. Не будем требовать отсутствия пустого и стартового в правой части.

Обобщение. Смотрим на транзитивное замыкание.

### 6.3 Алгоритм на основе матриц

Ссылка на Валианта [?].

Оригинальные матрицы (Рустам) [1]

Кратчайшие.

Одно дерево?

### 6.4 Конъюнктивные и булевы грамматики

#### 6.4.1 Определения

Охотин [?].

### 6.4.2 Для графов

Классическая семантика — работает для конъюнктивных для любых графов. Для Булевых и конъюнктивных только для графов без циклов.

Альтернативная семантика (DataLog). Трактуем конъюнкцию как в даталоге. Тогда всё хорошо. Это похоже на даталог через линейную алгебру [?]

## 6.5 Особенности реализации матричного алгоритма

Кое-что про наши реализации [7]

Разреженные матрицы, плотные матрицы, GraphBLAS<sup>1</sup>, GPGPU, CUTLASS<sup>2</sup>.

Расположение в памяти: хорошо, когда всё влезло на одну карту.

Распределённые решения. Через GraphBLAS

## 6.6 Обзор

Китайцы [?], Брэдфорд [?], Ли [5], Хеллингс [?], OpenCypher [4].

Рассуждения про асимптотику.

Субкубический для частного случая (Брэдфорд) [2].

## 6.7 Вопросы и задачи

1. !!!

## 7 Через тензорное произведение

Предыдущий подход позволяет выразить задачу поиска путей с ограничениями в терминах нормальных языков через набор матричных операций. Это позволяет использовать высокопроизводительные библиотеки и массовопараллельные архитектуры и вообще круто. Однако, такой подход требует, чтобы грамматика находилась в ослабленной нормальной форме Хомского, что приводит к её разрастанию. Можно ли как-то избежать этого?

В данном разделе мы попробуем предложить альтернативное сведение задачи поиска путей к матричным операциям. В результате мы сможем избежать преобразования грамматики в ОНФХ, однако, матрицы, с которыми нам придётся работать, будут существенно большего размера.

В основе подхода лежит использование рекурсивных сетей или рекурсивных автоматов в качестве представления контекстно-свободных грамматик.

---

<sup>1</sup>!!!

<sup>2</sup>Репозиторий библиотеки CUTLASS: <https://github.com/NVIDIA/cutlass>

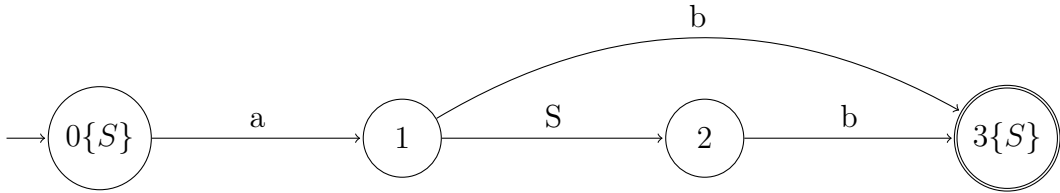
## 7.1 Рекурсивные автоматы и сети

Рекурсивный автомат или сеть — это представление контекстно-свободных грамматик, обобщающее конечные автоматы. В нашей работе мы будем придерживаться термина **рекурсивный автомат**. Классическое определение рекурсивного автомата выглядит следующим образом.

**Определение 7.1.** Рекурсивный автомат — это кортеж вида  $\langle N, \Sigma, S, D \rangle$ , где

- $N$  — нетерминальный алфавит;
- $\Sigma$  — терминальный алфавит;
- $S$  — стартовый нетерминал;
- $D$  — конечный автомат над  $N \cup \Sigma$  в котором стартовые и финальные состояния помечены подмножествами  $N$ .

Построим рекурсивный автомат для грамматики !!!!



Используем стандартные обозначения для стартовых и финальных состояний. Дополнительно в стартовых и финальных состояниях укажем нетерминалы, для которых эти состояния стартовые/финальные.

В некоторых случаях рекурсивный автомат можно рассматривать как конечный автомат над смешанным алфавитом. Именно такой взгляд мы будем использовать при изложении алгоритма.

## 7.2 Тензорное произведение

Тензорное произведение матриц или произведение Кронекера — это бинарная операция, обозначаемая  $\otimes$  и определяемая следующим образом.

**Определение 7.2.** Пусть даны две матрицы:  $A$  размера  $m \times n$  и  $B$  размера  $p \times q$ . Произведение Кронекера или тензорное произведение матриц  $A$  и  $B$  — это блочная матрица  $C$  размера  $mp \times nq$ , вычисляемая следующим образом:

$$C = A \otimes B = \begin{pmatrix} A_{0,0}B & \cdots & A_{0,n-1}B \\ \vdots & \ddots & \vdots \\ A_{m-1,0}B & \cdots & A_{m-1,n-1}B \end{pmatrix}$$

**Пример 7.1.**

$$\begin{aligned}
 \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \otimes \begin{pmatrix} 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} &= \begin{pmatrix} 1 \begin{pmatrix} 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} & 2 \begin{pmatrix} 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \\ 3 \begin{pmatrix} 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} & 4 \begin{pmatrix} 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \end{pmatrix} = \\
 &= \begin{pmatrix} 5 & 6 & 7 & 8 & 10 & 12 & 14 & 16 \\ 9 & 10 & 11 & 12 & 18 & 20 & 22 & 24 \\ 13 & 14 & 15 & 16 & 26 & 28 & 30 & 32 \\ 15 & 18 & 21 & 24 & 20 & 24 & 28 & 32 \\ 27 & 30 & 33 & 36 & 36 & 40 & 44 & 48 \\ 39 & 42 & 45 & 48 & 52 & 56 & 60 & 64 \end{pmatrix} \quad (1)
 \end{aligned}$$

Заметим, что для определения тензорного произведения матриц достаточно определить операцию умножения на элементах исходных матриц. Также отметим, что произведение Кронекера не является коммутативной. При этом всегда существуют такие две матрицы перестановок  $P$  и  $Q$ , что  $A \otimes B = P(B \otimes A)Q$ . Это свойство потребуется нам в дальнейшем.

Теперь перейдём к графам. Сперва дадим классическое определение тензорного произведения двух неориентированных графов.

**Определение 7.3.** Пусть даны два графа:  $\mathcal{G}_1 = \langle V_1, E_1 \rangle$  и  $\mathcal{G}_2 = \langle V_2, E_2 \rangle$ . Тензорным произведением этих графов будем называть граф  $\mathcal{G}_3 = \langle V_3, E_3 \rangle$ , где  $V_3 = V_1 \times V_2$ ,  $E_3 = \{((v_1, v_2), (u_1, u_2)) \mid (v_1, u_1) \in E_1 \text{ и } (v_2, u_2) \in E_2\}$ .

Иными словами, тензорным произведением двух графов является граф, такой что:

1. множество вершин — это прямое произведение множеств вершин исходных графов;
2. ребро между вершинами  $v = (v_1, v_2)$  и  $u = (u_1, u_2)$  существует тогда и только тогда, когда существуют рёбра между парами вершин  $v_1, u_1$  и  $v_2, u_2$  в соответствующих графах.

Для того, чтобы построить тензорное произведение ориентированных графов, необходимо в предыдущем определении, в условии существования ребра в результирующем графе, дополнительно потребовать, чтобы направления рёбер совпадали. Данное требование получается естественным образом, если считать, что пары вершин, задающие ребро упорядочены, поэтому формальное определение отличаться не будет.

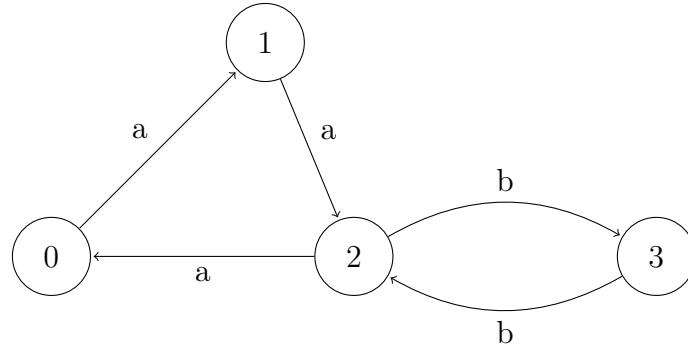
Осталось добавить метки к рёбрам. Это приведёт к логичному усилению требования к существованию ребра: метки рёбер в исходных графах должны совпадать. Таким образом, мы получаем следующее определение тензорного произведения ориентированных графов с метками на рёбрах.

**Определение 7.4.** Пусть даны два ориентированных графа с метками на рёбрах:  $\mathcal{G}_1 = \langle V_1, E_1, L_1 \rangle$  и  $\mathcal{G}_2 = \langle V_2, E_2, L_2 \rangle$ . Тензорным произведением этих графов будем называть граф  $\mathcal{G}_3 = \langle V_3, E_3, L_3 \rangle$ , где  $V_3 = V_1 \times V_2$ ,  $E_3 = \{((v_1, v_2), l, (u_1, u_2)) \mid (v_1, l, u_1) \in E_1 \text{ и } (v_2, l, u_2) \in E_2\}$ ,  $L_3 = L_1 \cap L_2$ .

Нетрудно заметить, что матрица смежности графа  $\mathcal{G}_3$  равна тензорному произведению матриц смежностей исходных графов  $\mathcal{G}_1$  и  $\mathcal{G}_2$ .

Рассмотрим пример. В качестве одного из графов возьмём рекурсивный автомат, построенный ранее!!!. Его матрица смежности выглядит следующим образом.

$$M_1 = \begin{pmatrix} \cdot & [a] & \cdot & \cdot \\ \cdot & \cdot & [S] & [b] \\ \cdot & \cdot & \cdot & [b] \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$



Второй граф представлен на изображении 7.2. Его матрица смежности имеет следующий вид.

$$M_2 = \begin{pmatrix} \cdot & [a] & \cdot & \cdot \\ \cdot & \cdot & [a] & \cdot \\ [a] & \cdot & \cdot & [b] \\ \cdot & \cdot & [b] & \cdot \end{pmatrix}$$

Теперь вычислим  $M_1 \otimes M_2$ .

$$M_3 = M_1 \otimes M_2 = \begin{pmatrix} \cdot & [a] & \cdot & \cdot \\ \cdot & \cdot & [S] & [b] \\ \cdot & \cdot & \cdot & [b] \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \otimes \begin{pmatrix} \cdot & [a] & \cdot & \cdot \\ \cdot & \cdot & [a] & \cdot \\ [a] & \cdot & \cdot & [b] \\ \cdot & \cdot & [b] & \cdot \end{pmatrix} =$$

$$= \left( \begin{array}{cccc|cccc|cccc|cccc} \cdot & \cdot & \cdot & \cdot & \cdot & [a] & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & [a] & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & [a] & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & [b] & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & [b] & \cdot & \cdot \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & [b] & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & [b] & \cdot & \cdot & \cdot \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{array} \right) \quad (2)$$

Побалуемся с некоммутативностью и перестановками.



## 7.3 Алгоритм

Идея алгоритма основана на обобщении пересечения двух конечных автоматов до пересечения рекурсивного автомата, построенного по грамматике, со входным графом.

Пересечение двух конечных автоматов — тензорное произведение соответствующих графов. Пересечение языков коммутативно, тензорное произведение нет, но это не страшно.

Несколько наблюдений. Путь из нетерминалов и терминалов. При этом должны быть пути из терминалов. Иначе не задать язык. Будем насыщать граф рёбрами, помеченными нетерминалами.

По грамматике строим автомат. г 13

---

**Listing 1** Поиск путей через тензорное произведение

---

```
1: function CONTEXTFREEPATHQUERYING( $G, \mathcal{G}$ )
2:    $N \leftarrow$  рекурсивный автомат для  $G$ 
3:    $M_1 \leftarrow$  матрица смежности  $N$ 
4:    $M_2 \leftarrow$  матрица смежности  $\mathcal{G}$ 
5:    $T \leftarrow$  the matrix  $n \times n$  in which each element is  $\emptyset$ 
6:   for all  $(i, x, j) \in E$  do                                      $\triangleright$  Matrix initialization
7:      $T_{i,j} \leftarrow T_{i,j} \cup \{A \mid (A \rightarrow x) \in P\}$ 
8:   while matrix  $T$  is changing do
9:      $M_3 \leftarrow M_1 \otimes M_2$                                       $\triangleright$  Graphs intersection
10:     $M_3 \leftarrow transitiveClosure(M_3)$ 
11:    for do
12:      for do
13:        if then
14:           $Nt \leftarrow M_3[i, j] \cup \{\}$ 
15:           $m \leftarrow M_3[i, j] \cup \{\}$ 
16:           $n \leftarrow M_3[i, j] \cup \{\}$ 
17:           $M_2[n, m] \leftarrow M_2[n, m] \cup \{Nt\}$ 
18:  return  $T$ 
```

---

В цикле: пересекли через тензорное произведение, замкнули через обычное матричное произведение, чтобы найти пути из начальной в конечную в грамматике, поставили в соответствующие ячейки нетерминалы, добавили соответствующие рёбра в исходный граф.

Можно вычислять только разницу. Для этого, правда, потребуется держать ещё одну матрицу. И надо проверять, что вычислительно дешевле: поддерживать разницу и потом каждый раз поэлементно складывать две матрицы или каждый раз вычислять полностью произведение.

Всего несколько матриц. Разреженные. Необходимо отметить, что для реальных графов и запросов результат тензорного произведения будет очень разрежен. На готовых либах должно быть быстро.

## 7.4 Примеры

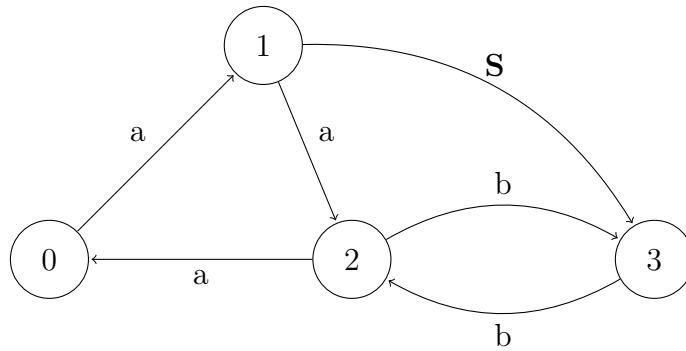
Рассмотрим подробно ряд примеров работы описанного алгоритма. Будем для каждой итерации внешнего цикла выписывать результаты основных операций: тензорного произведения, транзитивного замыкания, обновления матрицы смежности входного графа. Новые, по сравнению с предыдущим состоянием, элементы матриц будем выделять жирным.

**Пример 7.2.** Теоретический худший случай. Такой же как и для матричного.

**Итерация 1 (конец).** Начало в разделе!!!, где мы вычислили тензорное произведение матриц смежности. Теперь нам осталось только вычислить транзитивное замыкание полученной матрицы.

$$tc(M_3) = \left( \begin{array}{cccc|cccc|cccc|cccc} \cdot & \cdot & \cdot & \cdot & \cdot & [a] & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & [a] & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & [ab] \\ \cdot & \cdot & \cdot & \cdot & [a] & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & [b] \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & [b] & \cdot & \cdot \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \hline \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{array} \right) \quad (3)$$

Мы видим, что в результате транзитивного замыкания появилось новое ребро с меткой  $ab$  из вершины  $(0, 1)$  в вершину  $(3, 3)$ . Так как вершина 0 является стартовой в рекурсивном автомате, а 3 является финальной, то слово вдоль пути из вершины 1 в вершину 3 во входном графе выводимо из нетерминала  $S$ . Это означает, что в графе должно быть добавлено ребро из 0 в 3 с меткой  $S$ , после чего граф будет выглядеть следующим образом:



Матрица смежности обновлённого графа:

$$M_2 = \begin{pmatrix} \cdot & [a] & \cdot & \cdot \\ \cdot & \cdot & [a] & [S] \\ [a] & \cdot & \cdot & [b] \\ \cdot & \cdot & [b] & \cdot \end{pmatrix}$$

Итерация закончена. Возвращаемся к началу цикла и вновь вычисляем тензорное произведение.

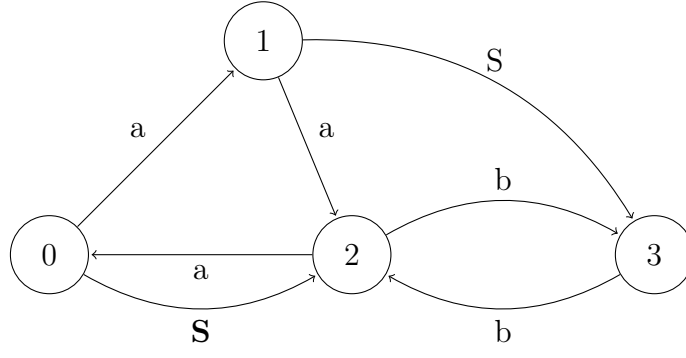
**Итерация 2.** Вычисляем тензорное произведение матриц смежности.

[illegible]

Вычисляем транзитивное замыкание полученной матрицы:

[illegible]

В транзитивном замыкании появилось три новых ребра, лжнако только одно из них соединяет вершины, соответствующие сартовому и конечному состоянию входного рекурсивного автомата. Только это ребро должно быть соответствующим образом добавлено во входной граф. В итоге, обновлённый граф:



И его матрица смежности:

$$M_2 = \begin{pmatrix} \cdot & [a] & [S] & \cdot \\ \cdot & \cdot & [a] & [S] \\ [a] & \cdot & \cdot & [b] \\ \cdot & \cdot & [b] & \cdot \end{pmatrix}$$

**Итерация 3.** Снова начинаем с тензорного произведения.

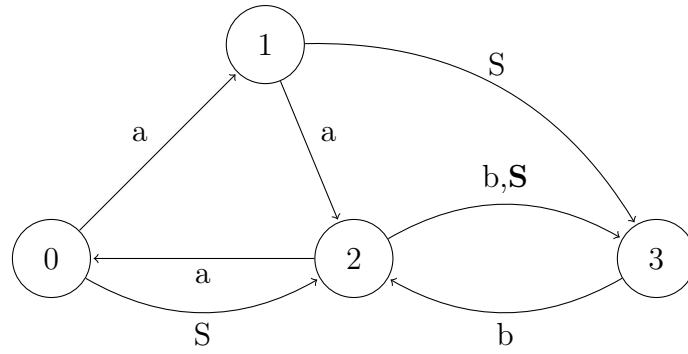
$$M_3 = M_1 \otimes M_2 = \begin{pmatrix} \cdot & [a] & \cdot & \cdot \\ \cdot & \cdot & [S] & [b] \\ \cdot & \cdot & \cdot & [b] \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \otimes \begin{pmatrix} \cdot & [a] & [S] & \cdot \\ \cdot & \cdot & [a] & [S] \\ [a] & \cdot & \cdot & [b] \\ \cdot & \cdot & [b] & \cdot \end{pmatrix} =$$

$$= \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & [a] & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & [a] & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & [a] & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \quad (6)$$

Затем вычисляем транзитивное замыкание:

$$tc(M_3) = \left( \begin{array}{c|c|c|c} \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & [a] & \cdot & \cdot \\ \cdot & \cdot & [a] & \cdot \\ [a] & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & \cdot & [aS] \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & [aS] & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & [aSb] & \cdot \\ \cdot & \cdot & \cdot & [ab] \\ \cdot & \cdot & \cdot & [aSb] \\ \cdot & \cdot & \cdot & \cdot \end{matrix} \\ \hline \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & [S] & \cdot \\ \cdot & \cdot & \cdot & [S] \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & \cdot & [Sb] \\ \cdot & \cdot & [Sb] & \cdot \\ \cdot & \cdot & \cdot & [b] \\ \cdot & \cdot & [b] & \cdot \end{matrix} \\ \hline \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & [b] \\ \cdot & \cdot & [b] & \cdot \end{matrix} \\ \hline \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} \end{array} \right) \quad (7)$$

И наконец обновляем граф:



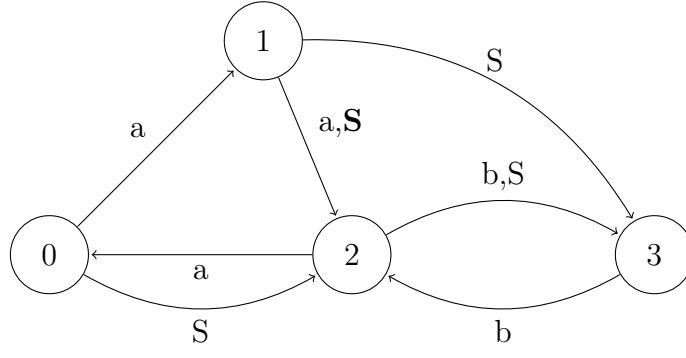
Матрица смежности обновлённого графа:

$$M_2 = \begin{pmatrix} \cdot & [a] & [S] & \cdot \\ \cdot & \cdot & [a] & [S] \\ [a] & \cdot & \cdot & [b, S] \\ \cdot & \cdot & [b] & \cdot \end{pmatrix}$$

Уже можно заметить закономерность: на каждой итерации мы добавляем ровно одно новое ребро во входной граф. То есть находим ровно одну новую пару вершин, между которыми существует интересующий нас путь. Попробуйте спрогнозировать, сколько итераций нам ещё осталось сделать.

**Итерация 4.** Продолжаем. Вычисляем тензорное произведение.





Матрица смежности обновлённого графа:

$$M_2 = \begin{pmatrix} \cdot & [a] & [S] & \cdot \\ \cdot & \cdot & [a, \mathbf{S}] & [S] \\ [a] & \cdot & \cdot & [b, S] \\ \cdot & \cdot & [b] & \cdot \end{pmatrix}$$

**Итерация 5.** Приступаем к выполнению следующей итерации основного цикла. Вычисляем тензорное произведение.

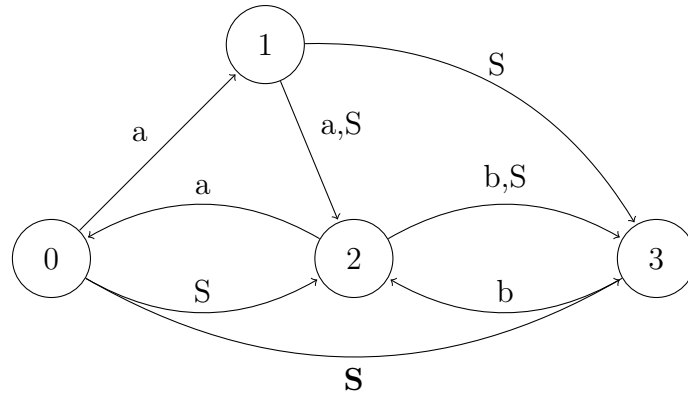
$$M_3 = M_1 \otimes M_2 = \begin{pmatrix} \cdot & [a] & \cdot & \cdot \\ \cdot & \cdot & [S] & [b] \\ \cdot & \cdot & \cdot & [b] \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \otimes \begin{pmatrix} \cdot & [a] & [S] & \cdot \\ \cdot & \cdot & [a, S] & [S] \\ [a] & \cdot & \cdot & [b, S] \\ \cdot & \cdot & [b] & \cdot \end{pmatrix} =$$

$$= \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & [a] & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & [a] & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & [a] & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \quad (10)$$

Затем вычисляем транзитивное замыкание:

$$tc(M_3) = \left( \begin{array}{c|c|c|c} \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & [a] & \cdot & \cdot \\ \cdot & \cdot & [a] & \cdot \\ [a] & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & [\mathbf{aS}] & [aS] \\ \cdot & \cdot & \cdot & [aS] \\ \cdot & \cdot & [aS] & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & [aSb] & [\mathbf{aSb}] \\ \cdot & \cdot & [aSb] & [ab] \\ \cdot & \cdot & \cdot & [aSb] \\ \cdot & \cdot & \cdot & \cdot \end{matrix} \\ \hline \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & [S] & \cdot \\ \cdot & \cdot & [S] & [S] \\ \cdot & \cdot & \cdot & [S] \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & \cdot & [Sb] \\ \cdot & \cdot & [Sb] & [\mathbf{Sb}] \\ \cdot & \cdot & [Sb] & [b] \\ \cdot & \cdot & [b] & \cdot \end{matrix} \\ \hline \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & [b] \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & [b] & \cdot \end{matrix} \\ \hline \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} & \begin{matrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{matrix} \end{array} \right) \quad (11)$$

Обновляем граф:



Матрица смежности обновлённого графа:

$$M_2 = \begin{pmatrix} \cdot & [a] & [S] & [\mathbf{S}] \\ \cdot & \cdot & [a, S] & [S] \\ [a] & \cdot & \cdot & [b, S] \\ \cdot & \cdot & [b] & \cdot \end{pmatrix}$$

**Итерация 6.** И наконец последняя содержательная итерация основного цикла.

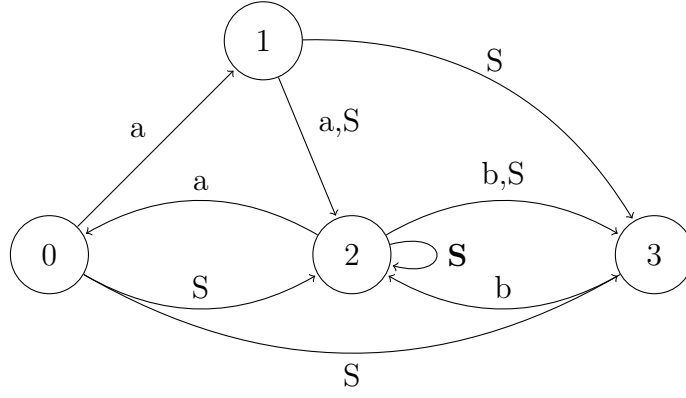


[illegible]

Транзитивное замыкание:

[illegible]

Обновлённый граф:



И матрица смежности:

$$M_2 = \begin{pmatrix} . & [a] & [S] & [S] \\ . & . & [a, S] & [S] \\ [a] & . & \mathbf{[S]} & [b, S] \\ . & . & [b] & . \end{pmatrix}$$

Следующая итерация не приведёт к изменению графа. Читатель может убедиться в этом самостоятельно. Соответственно алгоритм можно завершать. Нам потребовалось семь итераций (шесть содержательных и одна проверочная).

Матрица смежности получилась такая же, как и раньше, ответ правильный. Мы видим, что количество итераций внешнего цикла такое же как и у алгоритма !!! И ещё что-то видим и можем понять.

**Пример 7.3.** В данном примере мы увидим, как структура грамматики и, соответственно, рекурсивного автомата, влияет на процесс вычислений.

Интуитивно понятно, что чем меньше состояний в рекурсивной сети, тем лучше. То есть желательно получить как можно более компактное описание контекстно-свободного языка.

Для примера возьмём в качестве КС языка язык Дика на одном типе скобок и опишем его двумя различными грамматиками. Первая грамматика классическая:

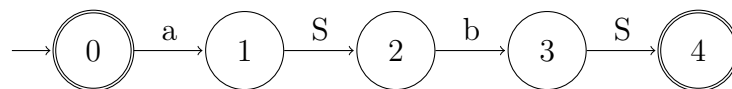
$$G_1 = \langle \{a, b\}, \{S\}, \{S \rightarrow a S b S \mid \varepsilon\} \rangle$$

Во второй грамматике мы будем использовать конструкции регулярных выражений в правой части правил. То есть вторая грамматика находится в EBNF [?].

$$G_2 = \langle \{a, b\}, \{S\}, \{S \rightarrow (a S b)^*\} \rangle$$

Построим рекурсивные автоматы  $N_1$  и  $N_2$  и их матрицы смежности для этих грамматик.

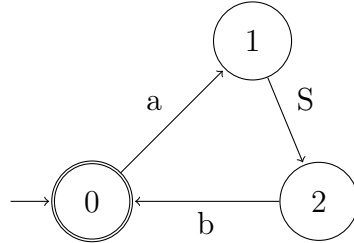
Рекурсивный автомат  $N_1$  для грамматики  $G_1$ :



Матрица смежности  $N_1$ :

$$M_1^1 = \begin{pmatrix} \cdot & [a] & \cdot & \cdot & \cdot \\ \cdot & \cdot & [S] & \cdot & \cdot \\ \cdot & \cdot & \cdot & [b] & \cdot \\ \cdot & \cdot & \cdot & \cdot & [S] \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

Рекурсивный автомат  $N_2$  для грамматики  $G_2$ :



Матрица смежности  $N_2$ :

$$M_1^2 = \begin{pmatrix} \cdot & [a] & \cdot \\ \cdot & \cdot & [S] \\ [b] & \cdot & \cdot \end{pmatrix}$$

Первое, очевидное, наблюдение — количество состояний в  $N_2$  меньше, чем в  $N_1$ . Это значит, что матрицы будут меньше, считать быстрее.

Для того, чтобы проще было сделать второе, сперва выполним пошагово алгоритм для двух заданных рекурсивных сетей.

Вход возьмём линейный:



Сразу дополним матрицу смежности нетерминалами, выводящими пустую строку, и получим следующую матрицу:

$$M_2 = \begin{pmatrix} [S] & [a] & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & [S] & [b] & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & [S] & [a] & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & [S] & [b] & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & [S] & [a] & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & [S] & [b] \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & [S] \end{pmatrix}$$

Сперва запустим алгоритм на входе и  $N_2$ . Первый шаг первой итерации — вычисление тензорного произведения  $M_1^2 \otimes M_2$ .

[illegible]

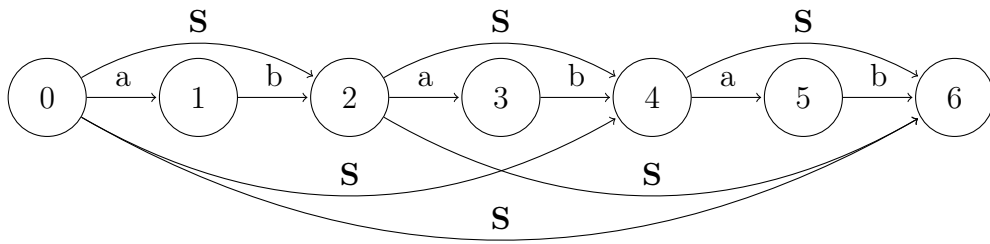
Опустим промежуточные шаги вычисления транзитивного замыкания  $M_3$  и сразу пред-

ставим конечный результат:

$$tc(M_3) = \begin{pmatrix} \begin{array}{cccc|cccc} \cdot & \cdot & [aSb] & \cdot & [aSbaSb] & \cdot & [aSbaSbaSb] & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & [aSb] & \cdot & [aSbaSb] & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & [aSb] & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{array} & \begin{array}{cccc|cccc} \cdot & [a] & \cdot & [aSba] & \cdot & [aSbaSba] & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & [a] & \cdot & [aSba] & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & [a] & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{array} & \begin{array}{cccc|cccc} \cdot & [aS] & \cdot & [aSbaS] & \cdot & [aSbaSbaS] & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & [aS] & \cdot & [aSbaS] & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & [aS] & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & [aS] \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{array} \end{pmatrix} \quad (15)$$

В результате вычисления транзитивного замыкания появилось большое количество новых рёбер, однако нас будут интересовать только те, информация о которых храниться в левом верхнем блоке. Остальные рёбра не соответствуют принимающим путям в рекурсивном автомате (убедитесь в этом самостоятельно).

После добавления соответствующих рёбер, мы получим следующий граф:



Его матрица смежности:

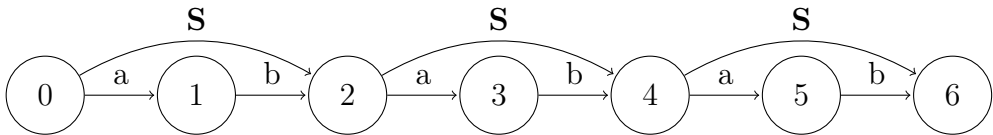
$$M_2 = \begin{pmatrix} [S] & [a] & [S] & \cdot & [S] & \cdot & [S] \\ \cdot & [S] & [b] & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & [S] & [a] & [S] & \cdot & [S] \\ \cdot & \cdot & \cdot & [S] & [b] & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & [S] & [a] & [S] \\ \cdot & \cdot & \cdot & \cdot & \cdot & [S] & [b] \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & [S] \end{pmatrix}$$

Теперь запустим алгоритм на второй грамматике и том же входе.

$$(16)$$

### Транзитивное замыкание.

## Обновлённый граф



Его матрица смежности:

$$M_2 = \begin{pmatrix} [S] & [a] & [\mathbf{S}] & \cdot & \cdot & \cdot & \cdot \\ \cdot & [S] & [b] & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & [S] & [a] & [\mathbf{S}] & \cdot & \cdot \\ \cdot & \cdot & \cdot & [S] & [b] & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & [S] & [a] & [\mathbf{S}] \\ \cdot & \cdot & \cdot & \cdot & \cdot & [S] & [b] \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & [S] \end{pmatrix}$$

Потребуется ещё одна итерация.



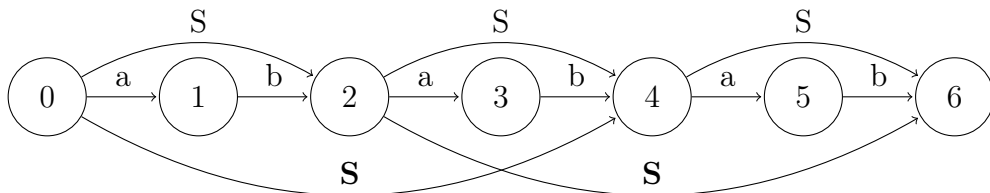
[illegible]

Транзитивное замыкание.

$$\begin{aligned}
& tc(M_3) = \\
& = \left( \begin{array}{c|c|c|c|c}
\begin{array}{c} \dots\dots\dots [a] \dots\dots\dots \\ \dots\dots\dots [a] \dots\dots\dots \\ \dots\dots\dots [a] \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \end{array} & \begin{array}{c} \dots [aS] \dots [aS] \dots \\ \dots \dots [aS] \dots [aS] \dots \\ \dots \dots \dots [aS] \dots \\ \dots \dots \dots \dots \\ \dots \dots \dots \dots \end{array} & \begin{array}{c} \dots [aSb] \dots [aSb] \dots \\ \dots \dots [aSb] \dots [aSb] \dots \\ \dots \dots \dots [aSb] \dots \\ \dots \dots \dots \dots \\ \dots \dots \dots \dots \end{array} & \begin{array}{c} \dots [aSbS] \dots [aSbS] \dots \\ \dots \dots \dots [aSbS] \dots [aSbS] \\ \dots \dots \dots \dots [aSbS] \\ \dots \dots \dots \dots \dots \\ \dots \dots \dots \dots \end{array} \\
\hline
\begin{array}{c} \dots\dots\dots [S] \dots [S] \dots \\ \dots\dots\dots [S] \dots [S] \dots \\ \dots\dots\dots [S] \dots [S] \dots \\ \dots\dots\dots [S] \dots [S] \dots \\ \dots\dots\dots [S] \dots [S] \dots \\ \dots\dots\dots [S] \dots [S] \dots \\ \dots\dots\dots [S] \dots [S] \dots \end{array} & \begin{array}{c} \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \end{array} & \begin{array}{c} \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \end{array} & \begin{array}{c} \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \end{array} \\
\hline
\begin{array}{c} \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \end{array} & \begin{array}{c} \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \end{array} & \begin{array}{c} \dots [b] \dots\dots\dots \\ \dots\dots\dots [b] \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \end{array} & \begin{array}{c} \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \end{array} \\
\hline
\begin{array}{c} \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \end{array} & \begin{array}{c} \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \end{array} & \begin{array}{c} \dots [S] \dots [S] \dots \\ \dots [S] \dots [S] \dots \\ \dots [S] \dots [S] \dots \\ \dots [S] \dots [S] \dots \\ \dots [S] \dots [S] \dots \\ \dots [S] \dots [S] \dots \\ \dots [S] \dots [S] \dots \end{array} & \begin{array}{c} \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \end{array} \\
\hline
\begin{array}{c} \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \end{array} & \begin{array}{c} \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \end{array} & \begin{array}{c} \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \end{array} & \begin{array}{c} \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \\ \dots\dots\dots \dots\dots\dots \end{array}
\end{array} \right)
\end{aligned}$$

(19)

Обновлённый граф. На этом шаге мы смогли “склеить” из подстрок, вводимых из  $S$ , более длинные пути. Однако, согласно правилам грамматики, мы смогли “склеить” только две подстроки в единое целое.



Матрица смежности обновлённого графа:

$$M_2 = \begin{pmatrix} [S] & [a] & [S] & \cdot & [\mathbf{S}] & \cdot & \cdot \\ \cdot & [S] & [b] & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & [S] & [a] & [S] & \cdot & [\mathbf{S}] \\ \cdot & \cdot & \cdot & [S] & [b] & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & [S] & [a] & [S] \\ \cdot & \cdot & \cdot & \cdot & \cdot & [S] & [b] \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & [S] \end{pmatrix}$$

И, наконец, последняя содержательная итерация.

[illegible]

Транзитивное замыкание.



## 7.5 Замечания о реализации

Блочная структура матриц даёт хорошую основу для распределённого умножения при построении транзитивного замыкания.

Кодирование данных в ячейках.

Транзитивное замыкание.

## 7.6 Вопросы и задачи

1. Оценить пространственную сложность алгоритма.
2. Оценить временную сложность алгоритма.
3. Найти библиотеку для тензорного произведения. Реализовать алгоритм. Можно предположить, что запросы содержат ограниченное число терминалов и нетерминалов. Провести замеры. Сравнить с матричным.
4. Реализовать распределённое решение. См. блочную структуру

## 8 Сжатое представление леса разбора

Матричный алгоритм даёт нам ответ на вопрос о достижимости, но не предоставляет самих путей. Что делать, если мы хотим построить все пути, удовлетворяющие ограничениям?

Проблема в том, что множество путей может быть бесконечным. Можем ли мы предложить конечную структуру, однозначно описывающую такое множество? Вспомним, что пересечение контекстно-свободного языка с регулярным — это контекстно-свободный язык. Мы знаем, что контекстно-свободный язык можно описать контекстно-свободной грамматикой, которая конечна. Это и есть решение нашего вопроса. Осталось только научиться строить такую грамматику.

### 8.1 Лес разбора как представление контекстно-свободной грамматики

Добавление информации в классическое дерево разбора. Координаты и промежуточные узлы.

### 8.2 Вопросы и задачи

1. Постройте дерево вывода цепочки  $w = aababb$  в грамматике  $G = \langle \{a, b\}, \{S\}, \{S \rightarrow \varepsilon \mid a S b S\}, S \rangle$ .
2. Постройте все левосторонние выводы цепочки  $w = ababab$  в грамматике  $G = \langle \{a, b\}, \{S\}, \{S \rightarrow \varepsilon \mid a S b \mid S S\}, S \rangle$ .
3. Постройте все правосторонние выводы цепочки  $w = ababab$  в грамматике  $G = \langle \{a, b\}, \{S\}, \{S \rightarrow \varepsilon \mid a S b \mid S S\}, S \rangle$ .

4. Постройте все деревья вывода цепочки  $w = ababab$  в грамматике  $G = \langle \{a, b\}, \{S\}, \{S \rightarrow \varepsilon \mid a S b \mid S S\}, S \rangle$ , соответствующие левосторонним выводам.
5. Постройте все деревья вывода цепочки  $w = ababab$  в грамматике  $G = \langle \{a, b\}, \{S\}, \{S \rightarrow \varepsilon \mid a S b \mid S S\}, S \rangle$ , соответствующие правосторонним выводам.
6. Как связаны между собой леса, полученные в предыдущих двух задачах (4 и 5)? Какие выводы можно сделать из такой связи?
7. Постройте сжатое представление леса разбора, полученного в задаче 4.
8. Постройте сжатое представление леса разбора, полученного в задаче 5.
9. Предъявите контекстно-свободную грамматику существенно неоднозначного языка. Возьмите цепочку длины больше пяти, прилежащую этому языку, и постройте все деревья вывода этой цепочки в предъявленной грамматике.
10. Постройте сжатое представление леса, полученного в задаче 9.

## 9 Алгоритм на основе восходящего анализа

Наша реализация [8]

### 9.1 Восходящий синтаксический анализ

Основы LR-анализа.

Таблицы, конфликты.

Пример автомата и таблиц.

### 9.2 Вопросы и задачи

1. Постройте автомат для грамматики
2. Постройте таблицу для автомата из задачи
3. В том числе дать неоднозначную грамматику
4. Запустить, построить деревья, стеки и т.д.

## 10 Алгоритм на основе нисходящего анализа

GLL [3]

Другие реализации [6]

## 10.1 Нисходящий синтаксический анализ

## 10.2 GLL для КС запросов

## 10.3 Вопросы и задачи

1. Задача 1
2. Задача 2

# 11 Комбинаторы для КС апросов

## 11.1 Парсер комбинаторы

Что это, с чем едят, плюсы, минусы. Про семантику, безопасность, левую рекурсию и т.д.

## 11.2 Комбинаторы для КС

Комбинаторы [9]

## 11.3 Вопросы и задачи

1. Задача 1
2. Задача 2

# 12 От CFPQ к вычислению Datalog-запросов

## 12.1 Datalog

Конечные Эрбрановы модели. Наименьшая неподвижная точка.  $C :- d$

## 12.2 КС-запрос как запрос на Datalog

Покажем, что для данного графа и КС-запроса можно построить эквивалентный запрос на Datalog.

Пусть дан граф Граф преобразуется в набор фактов (базу данных).

Пусть есть граммтика  $G: S \rightarrow a b \mid a S b$ . Она может быть преобразована в запрос следующего вида.  $s(X, Y) :- a(X, Z), b(Z, Y)$ .  $s(X, Y) :- a(X, Z), s(Z, W)b(W, Y)$ .  $? :- s(X, Y)$

Наблюдения: появились переменные, есть порядок у конъюнктов, который задаёт порядок связывания.

## 12.3 Обобщение GLL для вычисления Datalog-запросов

Дескриптор — состояние процесса: состояние автомата, результат проделанной работы, подстановка. Задача — найти подстановки. На каждом шаге есть набор подстановок.



## Список литературы

- [1] R. Azimov and S. Grigorev. Context-free path querying by matrix multiplication. In *Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, GRADES-NDA '18, pages 5:1–5:10, New York, NY, USA, 2018. ACM.
- [2] P. G. Bradford. Efficient exact paths for dyck and semi-dyck labeled path reachability (extended abstract). In *2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, pages 247–253, Oct 2017.
- [3] S. Grigorev and A. Ragozina. Context-free path querying with structural representation of result. In *Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia, CEE-SECR '17*, pages 10:1–10:7, New York, NY, USA, 2017. ACM.
- [4] J. Kuijpers, G. Fletcher, N. Yakovets, and T. Lindaaker. An experimental study of context-free path query evaluation methods. In *Proceedings of the 31st International Conference on Scientific and Statistical Database Management, SSDBM '19*, pages 121–132, New York, NY, USA, 2019. ACM.
- [5] L. Lee. Fast context-free grammar parsing requires fast boolean matrix multiplication. *J. ACM*, 49(1):1–15, Jan. 2002.
- [6] C. M. Medeiros, M. A. Musicante, and U. S. Costa. Ll-based query answering over rdf databases. *Journal of Computer Languages*, 51:75 – 87, 2019.
- [7] N. Mishin, I. Sokolov, E. Spirin, V. Kutuev, E. Nemchinov, S. Gorbatyuk, and S. Grigorev. Evaluation of the context-free path querying algorithm based on matrix multiplication. In *Proceedings of the 2Nd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, GRADES-NDA'19, pages 12:1–12:5, New York, NY, USA, 2019. ACM.
- [8] E. Verbitskaia, S. Grigorev, and D. Avdyukhin. Relaxed parsing of regular approximations of string-embedded languages. In M. Mazzara and A. Voronkov, editors, *Perspectives of System Informatics*, pages 291–302, Cham, 2016. Springer International Publishing.
- [9] E. Verbitskaia, I. Kirillov, I. Nozkin, and S. Grigorev. Parser combinators for context-free path querying. In *Proceedings of the 9th ACM SIGPLAN International Symposium on Scala*, Scala 2018, pages 13–23, New York, NY, USA, 2018. ACM.