Министерство ОБЩЕГО И ПРОФЕССИОНАЛЬНОГО Образования
Российской Федерации
Санкт-Петербургский Государственный
университет информационных технологий, точной Механики и Оптики
Факультет информационных технологий и программирования

-

Шеметова Екатерина Николаевна

Задача поиска путей с контекстно-свободными ограничениями

Магистерская диссертация

Допущена к защите.
Зав. кафедрой:
д. ф.-м. н., профессор Омельченко А. В.

Научный руководитель:
к. ф.-м. н., доцент Григорьев С. В.

Рецензент:
ст. преп. ?

Санкт-Петербург
2019

ITMO UNIVERSITY
Information Technologies and Programming Faculty

Department of Mathematics and Information Technology

Ekaterina Shemetova

Context-Free Path Queries

Graduation Thesis

Admitted for defence.
Head of the chair:
professor Alexander Omelchenko

Scientific supervisor:
Semyon Grigorev

Reviewer:
assistant ?

Saint-Petersburg
2019

Оглавление

Введение

Графовая модель данных широко используется для решения задач, данные в которых тесно связаны между собой отношениями. Эти отношения как правило составляют сложную иерархию. Примеры сфер использования графовых моделей данных — графовые базы данных [4], RDF-графы [9], биоинформатика [29], моделирование и анализ социальных сетей [43], [7], статический анализ кода [42],[30],[41] и многое другое.

Так как роль графов — представить отношения между различными объектами, часто бывает необходимо вычислять запросы к большим графам с целью выявления сложных зависимостей между их вершинами. Естественным способом описать отношения между графовыми объектами является использование формальной грамматики над метками рёбер графа. Популярными являются запросы, использующие регулярные грамматики [22], [21], [31]. Также существуют запросы с использованием контекстно-свободных (КС) грамматик [15], [16], [2]. В данной работе будут рассмотренны запросы с использованием контекстно-свободных грамматик, так как такие запросы позволяют искать более сложные отношения по сравнению с запросами, использующими регулярные грамматики.

Таким образом, запросом является контекстно-свободная грамматика G, а результатом выполнения — множество всех троек (A, i, j), для которых путь из вершины i в вершину j таков, что строка, полученная конкатенацией меток рёбер этого пути, выводима из нетерминала A в грамматике G. Существующие алгоритмы для решения задачи с использованием реляционной семантики работают очень медленно в общем случае, при этом практически не изучены частные случаи (например, фиксированный подкласс графов), для которых возможно получить гораздо лучшее время работы. Как пример подобного подхода, в данной работе будет представлен алгоритм для решения задачи поиска путей с контекстно-свободными ограничениями с использованием реляционной семантики для ациклических графов. Также не изучена возможность эффективного распараллеливания данной задачи, поэтому в работе будет построена булева схема для её решения, дана оценка её глубины для общего случая и некоторых классов контекстно-свободных грамматик.

1. Описание предметной области

1.1. Синтаксический анализ

Синтаксический анализ — это процесс сопоставления линейной последовательности лексем (слов, токенов) языка с его формальной грамматикой [44]. Формальная грамматика или просто грамматика в теории формальных языков — способ описания формального языка, то есть выделения некоторого подмножества из множества всех слов некоторого конечного алфавита. Регулярные грамматики определяют в точности все регулярные языки, и поэтому эквивалентны конечным автоматам и регулярным выражениям. Контекстно-свободная грамматика G — это четверка $G = (N, \Sigma, P, S)$, где N — множество нетерминалов, Σ — терминалов, P — множество правил грамматики, S — стартовый нетерминал. *Деревом разбора* грамматики называется дерево, в вершинах которого записаны терминалы или нетерминалы. Все вершины, помеченные терминалами, являются листьями. Все вершины, помеченные нетерминалами, имеют детей. Дети вершины, в которой записан нетерминал, соответствуют раскрытию нетерминала по одному любому правилу (в левой части которого стоит этот нетерминал) и упорядочены так же, как в правой части этого правила. Контекстно-свободная грамматика для КС-языка без ε (пустая строка) [45] находится в *нормальной форме Хомского*, когда каждое из ее правил имеет одну из следующих форм:

- $A \to BC$
- $A \to a$, где $A, B, C \in N, a \in \Sigma$.

Контекстно-свободная грамматика находится в *нормальной форме Грейбах* тогда, когда её правила имеют вид:

- $A \to a\alpha$, где $\alpha \in (N \cup \Sigma)^*, a \in \Sigma$.

Важным подклассом контекстно-свободных язков являются языки Дика. Языки Дика — языки правильных скобочных последовательностей, формально правила грамматики языка Дика имеют следующий вид: $P = \{S \to \varepsilon, S \to SS, S \to a_i S a_i^{-1}\}$. Существуют также полу-Диковы языки, правила грамматики которых имеют вид $P = \{S \to \varepsilon, S \to SS, S \to a_i S a_i^{-1}, S \to a_i^{-1} S a_i\}$.

Существуют также булевы и конъюнктивные грамматики. Конъюнктивная грамматика является классом формальных грамматик, расширяющим класс контекстно-свободных грамматик с помощью булевой операции конъюнкции [25]. Булева грамматика является классом формальных грамматик, расширяющим класс контекстно-свободных грамматик с помощью булевых операций конъюнкции и отрицания [23].
Булева грамматика формально определена следующим образом:
$G = (\Sigma, N, R, S)$, где G — булева грамматика, Σ — терминальный алфавит, N —

нетерминальный алфавит (множество нетерминальных символов $\{A_1, A_2, ..., A_n\}$), S — стартовый нетерминал, R — множество правил грамматики.
Тогда правила булевой грамматики можно представить в виде:

$$A \to \alpha_1 \wedge \alpha_1 \wedge ... \wedge \alpha_m \wedge \neg\beta_1 \wedge \neg\beta_2 \wedge ... \wedge \neg\beta_n, \tag{1}$$

где A — нетерминал, $m + n \geq 1$, $\alpha_1, ..., \alpha_m, \beta_1, \beta_2, ..., \beta_n$ — строки, порождённые алфавитами Σ и N.

1.2. Поиск путей с контекстно-свободными ограничениями

Для начала определим задачу синтаксического анализа графа (поиска путей с контекстно-свободными ограничениями) с использованием реляционной семантики запросов. Рассмотрим ориентированный граф $D = (V, E)$, где V — множество вершин графа, а E — множество ребер и формальную грамматику G. Пусть у каждого ребра графа есть метка, множество всех меток обозначим Σ. Тогда каждый путь в D будет обозначать слово над алфавитом из Σ, полученное конкатенацией меток рёбер, включенных в этот путь. Для графа D и формальной грамматики $G = (N, \Sigma, P, S)$, мы обозначим отношения $R_A \subseteq V \times V$ для каждого $A \in N$ следующим образом:

$$R_A = \{(n, m) | \exists n\pi m(l(\pi) \in L(G_A))\} \tag{2}$$

, где n, m — вершины графа D, $n\pi m$ — путь из вершины n в вершину m, $l(\pi)$ — слово, полученное конкатенацией меток рёбер, принадлежащих пути π, а $L(G_A)$ - язык, порожденный грамматикой G со стартовым нетерминалом A.

Таким образом, задача синтаксического анализа графа D с использованием реляционной семантики запросов и формальной грамматики G сводится к нахождению всех троек (A, n, m), для которых путь $n\pi m$ таков, что строка $l(\pi)$ выводима из нетерминала A в грамматике G, то есть вычислению всех отношений R_A для любого $A \in N$.

Также задачу можно эквивалентно определить как задачу перечисления кратчайших строк, выводимых из всех нетерминалов грамматики языка $\mathcal{L}$, являющегося пересечением входной контекстно-свободной грамматики и регулярного языка, описывающего граф D. Формально, для каждой пары вершин $n, m \in D$ и нетерминала $A \in N$ $\mathcal{L} = \mathcal{L}(G; A) \cap \mathcal{L}(D; n, m)$, где $\mathcal{L}(G; A)$ — язык, описываемый входной контекстно-свободной грамматикой G со стартовым нетерминалом A, а $\mathcal{L}(D; n, m)$ — регулярный язык, описывющий все пути из вершины n в m.

1.3. Модели параллельных вычислений

Перед тем как создавать параллельные алгоритмы и анализировать их сложность, выбирается модель параллельных вычислений. *Модель параллельных вычислений* —

параметризованное описание класса машин [13]. В литературе предложен обширный ряд моделей параллельных вычислений, например модели с разделяемой памятью (PRAM, k-PRAM, SIMDAG и другие), булевы схемы, модели с распределенной памятью и другие. В данной работе будут рассмотрена модель булевых схем и модель SIMDAG.

Parallel Random Access Machine (PRAM) — модель параллельных вычислений, состоящая из неограниченного набора RAM-процессоров $P_0, P_1, P_2, ...$ и неограниченного набора разделяемых ячеек памяти $C_0, C_1, C_2, ...$ Каждый процессор в данной модели имеет свою собственную локальную память, знает свой индекс i и имеет инструкции для прямых и непрямых записей/чтений в разделяемую память. *The Single Instruction stream Multiple DAta stream Global memory machine(SIMDAG)* — разновидность модели PRAM, где счетчик команд (регистр PC) общий для всех процессоров, например, все активные процессоры исполняют одну и ту же инструкцию π_i на каждом шаге вычислений. Каждый процессор может оперировать над значениями данных, отличающимися от данных, с которыми оперируют другие процессоры. Различают unit-cost SIMDAG и log-cost SIMDAG. В модели unit-cost SIMDAG инструкция выполняется за единичное время, в модели log-cost SIMDAG время выполнения инструкции пропорционально числу бит, необходимых, чтобы представить данные и адреса, относящиеся к инструкции.

Пусть машина M — PRAM. Вход $x \in \{0,1\}^n$ представлен в M с помощью помещения числа n в ячейку разделяемой памяти C_0 и биты $x_1, x_2, ..., x_n$ в ячейки разделяемой памяти $C_1, C_2, ..., C_n$. M отображает результат вычисления $y \in \{0,1\}^m$ похожим образом: числом m в ячейке разделяемой памяти C_0 и битами $y_1, y_2, ..., y_m$ в ячейках разделяемой памяти $C_{1,\,2}, ..., C_m$.

M вычисляет за параллельное время $t(n)$ с помощью $p(n)$ процессоров, если для каждого входа $x \in \{0,1\}^n$ машина M остановится не более чем, через время $t(n)$, задействует не более $p(n)$ процессоров, и отобразит вывод $y \in \{0,1\}^m$.

Тогда эффективность вычислений можно оценить следующим образом.
Пусть f — функция из $\{0,1\}^*$ в $\{0,1\}^*$. Функция f вычисляется за параллельное время $t(n)$ с помощью $p(n)$ процессоров, если существует PRAM M, которая для входа x выдаёт на выходе $f(x)$ за время за параллельное время $t(n)$ с помощью $p(n)$ процессоров.

Модели булевых схем является математической моделью для описания контактных схем — реальных электротехнических устройств.
Пусть $B_k = \{f \mid f : \{0,1\}^k \to \{0,1\}\}$ — набор логических функций (отрицание, дизъюнкция, конъюнкция) с k аргументами. Тогда определим схему следующим образом. *Булева (логическая) схема* α — помеченный конечный ориентированный ациклический граф. Каждая вершина v имеет тип $\tau(v) \in I \cup B_0 \cup B_1 \cup B_2$. Вершины типа I называются *входом схемы* и имеют входную степень, равную нулю. Входы для α —

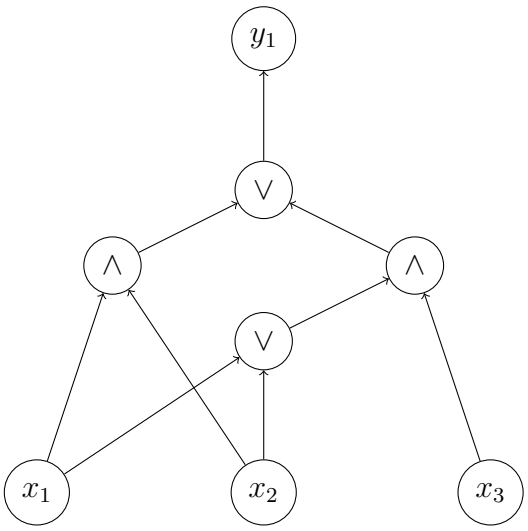

Рис. 1: Булева схема, вычисляющая функцию MAJ_3

это кортеж $\langle x_1, ..., x_n \rangle$ разных вершин. Вершина v с исходящей степенью 0 называется *выходом схемы*. Выходы для α — это кортеж $\langle y_1, ..., y_n \rangle$ разных вершин. Вершина v с $\tau(v) \in B_i$ имеет входную степень i и называется *гейтом*.
На рис. 1 представлена схема для вычисления функции MAJ_3, которая равна 1 тогда и только тогда, когда хотя бы две из ее переменных равны 1. Кортеж $\langle x_1, x_2, x_3 \rangle$ — вход схемы, y_1 — её единственный выход.

Булева схема α со входами $\langle x_1, ..., x_n \rangle$ и выходами $\langle y_1, ..., y_m \rangle$ вычисляет функцию $f : \{0,1\}^n \to \{0,1\}^m$ следующим способом: входу x_i приписывается значение $\nu(x_i)$ из $\{0,1\}$, представляющее i-ый аргумент фунции (i-ый бит входного значения). Каждой другой вершине приписывается уникальное значение $\nu(v) \in \{0,1\}$, полученное применением $\tau(v)$ к значениям вершин, рёбра из которых идут в v. Значение функции это кортеж $\langle \nu(y_1), ..., \nu(y_m) \rangle$, в котором каждый выход y_j соответствует j-му биту выходного значения.

Важными параметрами для оценки эффективности схемы являются её размер и глубина. *Размер схемы* α — число вершин в α. *Глубина схемы* α — длина самого длинного пути от входной вершины до выходной. Например, на глубина схемы на рис. 1 равна 3, размер — 8.

С моделью булевых схем связывают важные классы сложности. Для любого положительного целого числа i и входа размером n существует класс сложности NC^i, состоящий из булевых схем полиномиального от n размера, глубиной $O(\log^i n)$, использующая OR, AND и NOT гейты с ограниченной входной степенью. Если входная степень гейтов не ограничена, то говорят о классах AC^i.

1.4. Существующие решения

Существует ряд алгоритмов для синтаксического анализа графов с использованием реляционной семантики запросов и контекстно-свободных грамматик [16], [2], которые имеют низкую производительность на больших графах, так как их время работы в худшем случае порядка $O(n^5)$ для графа на n вершинах. Известно, что существование алгоритма с субкубическим временем работы является открытой проблемой. В работе [2] представлен алгоритм синтаксического анализа графов, использующий реляционную семантику запросов и КС-грамматики и вычисляющий матричное транзитивное замыкание с применением матричных операций, который позволяет использовать вычисления на графическом процессоре. Есть эффективный алгоритм, решающий данную задачу для Диковых и полу-Диковых языков на одном типе скобок, время работы которого составляет $O(n^{\omega}\log n^3)$, где n — число вершин в графе, а n^{ω} — время умножения булевых матриц размером $n \times n$ [5]. Также в работе [3] представлен алгоритм, решающий задачу для конъюнктивных грамматик.

Параллельных алгоритмов, решающих задачу с использованием реляционной семантики на сегодняшний день не представлено. Очевидно, что задача не менее сложная, чем определение принадлежности строки w контекстно-свободному яыку L. Для решения данной задачи имеются как булевы схемы, так и алгоритмы для других моделей параллельных вычислений [6], [35], [36]. Верхняя оценка для такой задачи — глубина схемы (параллельное время) $O(\log^2 n)$, где n — длина w.

Если рассматривать задачу как задачу пересечения регулярного языка с контекстно-свободным, то существует ряд булевых схем, решающих задачу проверки пустоты такого пересечения [8], [34], [37]. Однако, рассматриваемая в данной работе задача состоит в перечислении такого пересечения, что не является задачей разрешимости как задача выше, поэтому требуется дополнительное исследование. Также есть оценки сложности для похожей задачи: фиксирована пара вершин s и t в графе, помеченном символами из языка L, и нужно ответить на вопрос, существует ли в графе путь из вершины s в t, такой, что слово, полученное конкатенацией меток ребер этого пути, принадлежит языку L (задача L-Reachability) [20].

1.5. Постановка задачи

Как уже было сказано, алгоритмов, решающих задачу поиска путей с контекстно-свободными ограничениями эффективно, пусть даже на ограниченных подклассах графов, не представлено. Представленные выше алгоритмы работают с графами произвольной структуры. Если же исходный граф является ациклическим (деревом), можно воспользоваться известными свойствами ациклических графов, чтобы предоставить более производительный алгоритм для решения задачи синтаксического анализа для данной структуры графа. Например, многие эффективные алгоритмы син-

таксического анализа ([39], [18], [24], [12], [40]) являются алгоритмами динамического программирования, где вычисления производятся отдельно для каждой подстроки. Корректность таких вычислений обеспечивается тем, что на символах строки задан порядок. В случае синтаксического анализа графа данные алгоритмы использовать затруднительно из-за нелинейного порядка на вершинах графа. По тем же причинам на таком входе нельзя использовать стратегию “разделяй-и-властвуй“. Но известно, что на деревьях и ациклических графах порядок можно задать с помощью топологической сортировки. Так что одной из задач данной работы будет разработка и оценка алгоритма для синтаксического парсинга ациклических графов.

Так как параллельных алгоритмов для решения задачи не представлено, в данной работе будет построена булева схема, решающую задачу и оценены её параметры. Модель булевых схем удобна для оценки эффективности по сравнению с моделью PRAM, так как для нее существует большее количество классов сложности (NC^i). Также интерес представляет зависимость эффективности булевой схемы от контекстно-свободной грамматики, для которой она построена, так как существуют КС-языки, распознавание которых с помощью схемы можно осуществить быстрее, чем в общем случае для произвольной контекстно-свободной грамматики [14], [11], [17] — будет ли такое преимущество существовать в случае синтаксического анализа графов?

2. Синтаксический анализ ациклических графов

В работе [24] предложен быстрый алгоритм синтаксического анализа, основанный на матричных операциях и обобщённый для булевых грамматик. С помощью булевых грамматик можно выразить более широкий класс языков, а значит, формулировать более сложные запросы к графам. В данном разделе будет предложено расширение этого алгоритма для решения задачи синтаксического анализа ациклических графов и показана его корректность.

2.1. Аппроксимация решения

Рассматриваемый ниже алгоритм основан на матричных операциях. Так как необходимо найти тройки (A, i, j) для всех $A \in N$, результатом работы алгоритма на графе с n вершинами будет матрица размером $n \times n$, в каждой из ячеек которой будут содержаться подмножества нетерминалов. На каждой итерации алгоритма, подмножества нетерминалов могут объединяться с помощью операции конъюнкции, если во входной грамматике существует соответствующее правило. Поэтому важно объединять только те нетерминалы, которые соответствуют одному и тому же уникальному пути между парой вершин графа. В случае, если входной ациклический граф является деревом, между любой парой вершин графа будет существовать всего один путь. Если же граф не является деревом, то между одной парой вершин может быть в худшем случае 2^{n-2} путей, поэтому хранить и обрабатывать данные для каждого пути отдельно неэффективно. Так как информацию о всех путях из вершины i в вершину j эффективно хранить в одной ячейке, то с помощью конъюнкции могут быть объединены нетерминалы, соответствующие разным путям из i в j, но при этом будут объединены также нетерминалы, соответствующие одному и тому же пути, что и требуется для решения задачи. То же самое работает и для отрицаний в правиле: получив множества пар нетерминалов для одной пары вершин, необходимо проверить отсутствие в этом множестве определенных пар нетерминалов (согласно правилам грамматики). Так как неизвестно, какому конкретно пути соответствуют найденные нетерминалы, отрицание нельзя применять ко всему найденному множеству. Например, пусть в грамматике есть правило $A \to AB \wedge BC \wedge \neg DC$, а найденное множество пар нетерминалов для пары вершин (i, j) — $\{AB, BC, DC\}$. Если соотнести правило со всем множеством, то будет сделан вывод, что нетерминал A не может выводить строку $l(\pi)$ для какого-то пути $i\pi j$. Но между вершинами i и j могут существовать несколько путей, при этом один из них может выводиться множеством пар нетерминалов $\{AB, BC\}$, а другой — множеством $\{AB, DC\}$, например. Тогда выходит, что на самом деле $(i, j) \in R_A$. Поэтому, чтобы найти все булевы отношения R_A, необходимо построить аппроксимацию решения — рассмотреть все подмножества множества пар нетерминалов, тогда будут найдены все необходимые отношения плюс какие-то

дополнительные (для примера выше, таким дополнительным ответом являлось бы утверждение $(i, j) \in R_A$, если бы множество $\{AB, BC, DC\}$ соответствовало строго одному и тому же пути из вершины i в j.)

Поэтому приведенный ниже алгоритм будет вычислять аппроксимацию сверху для отношений R_A (надмножество необходимого множества).

2.2. Описание алгоритма

Пусть n - число вершин в графе D, вершины графа пронумерованы. $G = (N, \Sigma, S, R)$ — булева грамматика в нормальной форме.
Алгоритм использует следующие структуры данных:

1. Таблица парсинга T размером $n \times n$, представляющая собой верхнетреугольную матрицу, где для каждый элемент $T_{i,j}$ является множеством нетерминалов.
   Пусть $X \in (2^N)^{m \times l}$ и $Y \in (2^N)^{l \times n}$ – матрицы подмножеств множества нетерминалов N, а m, l, n – натуральные числа, большие единицы. Тогда зададим произведение матриц $X \times Y$ как матрицу $Z \in (2^N)^{m \times n}$, такую, что каждый её элемент $Z_{i,j}$ вычисляется по следующей формуле:

$$Z_{i,j} = \bigcup_{k=1}^{l} X_{i,k} \times Y_{k,j} \tag{3}$$

2. Таблица P, элементы $P_{i,j}$ которой принадлежат множеству пар нетерминалов $N \times N$.
   Используя значения $P_{i,j}$, можно получить набор нетерминалов для таблицы T — объединить пары нетерминалов в конъюнкты согласно правилам грамматики.

$$T_{i,j} = f(P_{i,j}), \tag{4}$$

   где $f : 2^{N \times N} \to 2^N$ для булевых грамматик определяется по формуле

$$f(P) = \bigcup_{k=1}^{2^{|P|}} \{A | \exists A \to B_1 C_1 \wedge ... \wedge B_m C_m \wedge \neg D_1 E_1 \wedge ... \wedge D'_m E'_m \in R : (B_t, C_t) \in P^k, (D_t, E_t) \notin P^k \forall t\}$$

   , где P^k — подмножество множества P. Функция f обеспечивает аппроксимацию сверху для операции отрицания: благодаря тому, что рассматриваются все подмножества нетерминалов, в результате не будут отсечены нужные отношения, например $P_{i,j} = \{AB, BC, CD\}$, и в грамматике есть правило $S \to AB \wedge BC \wedge \neg CD$. При этом, если в графе $AB \wedge BC$ соответствует одному пути из i в j, а CD — другому, то $(i, j) \in R_S$. В итоге для $P^k = \{AB, BC\}$ функция f вернёт S.

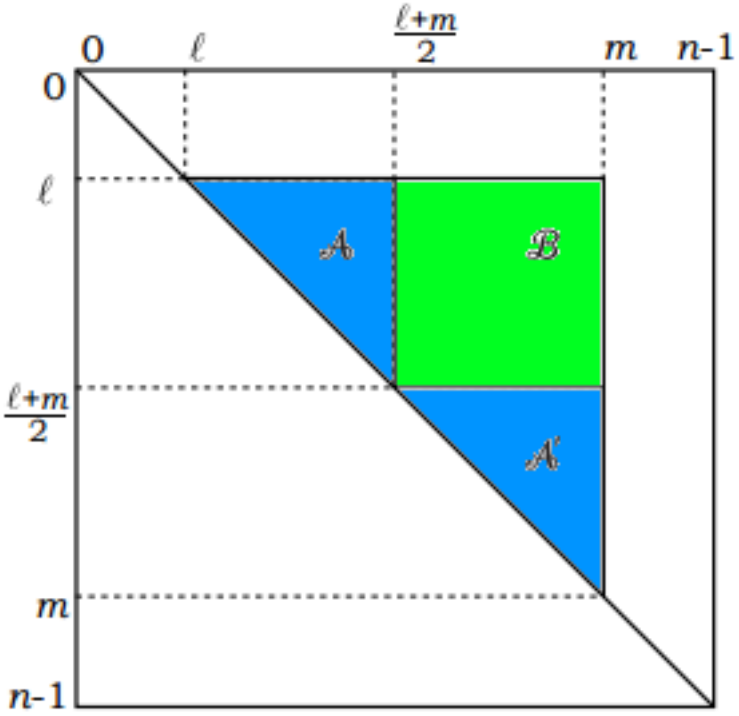

Рис. 2: Расположение подматриц матрицы T для вычисления произведений

В алгоритме Охотина элементы матрицы P рассчитываются группами с помощью выше определенного произведения подматриц (3) из таблицы T, дающих аналогичный результат с поэлементным умножением. Схема расположения подматриц представлена на рис. 2, 3.
Алгоритм состоит из двух рекурсивных процедур:

1. $compute(l, m)$ — рассчитывает значения $T_{i,j}$ для любых $l \leq i < j < m$.

2. $complete(l, m, l', m')$ — определена для $l \leq m < l' \leq m'$ при $m-l = m'-l'$ и $m - l$ без потери общности являющимися степенью двойки.
   Четыре входных параметра процедуры обозначают координаты подматрицы матрицы T, содержащей все элементы $T_{i,j}$, где $l \leq i < j < m$ и $l' \leq j < m'$. Они обозначают пути в графе, номер начала которых лежит между l и m, а конец между l' и m'. Так как ранее уже посчитаны $T_{i,j}$ для $l \leq i < j < m$ и $l' \leq i < j < m'$, а также $P_{i,j}$ для $l \leq i < m$ и $l' \leq j < m'$, процедура $complete(l, m, l', m')$ получает значения $T_{i,j}$ для $l \leq i < m$ и $l' \leq j < m'$.

Так как основная структура данных, с которой работает алгоритм, является верхнетреугольными матрицами, то тогда должен быть задан линейный порядок на вершинах графа D, такой, что любое ребро ведет от вершины с меньшим номером к вершине с большим номером. Если D является ациклическим графом, то данного свойства легко добиться с помощью топологической сортировки графа. Например,

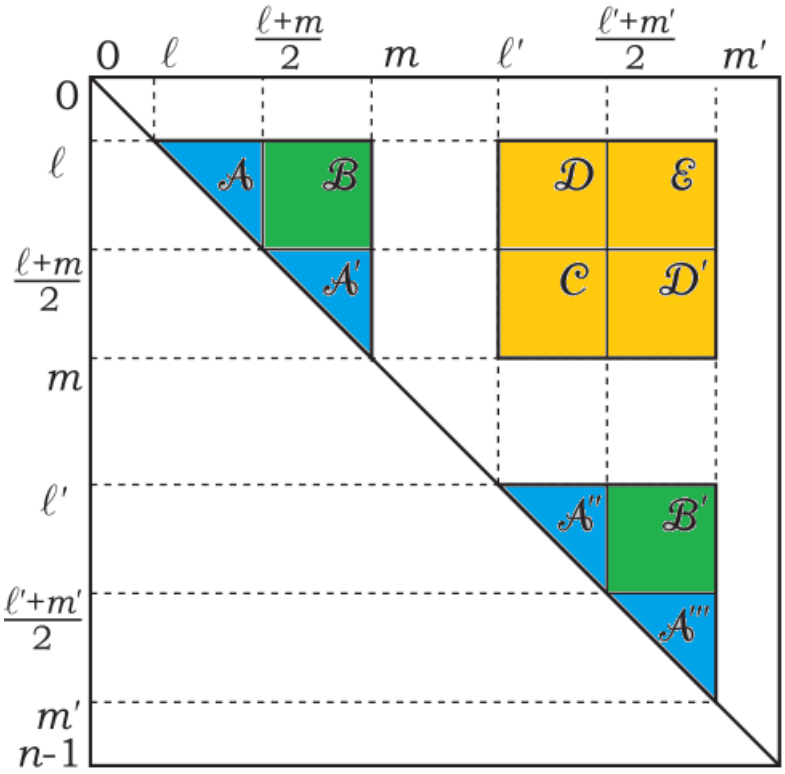

Рис. 3: Расположение подматриц матрицы T для вычисления произведений

алгоритм Тарьяна [38] позволяет осуществлять топологическую сортировку за линейное от количества вершин графа время $O(n)$.
Полный псевдокод алгоритма приведен в листинге 1.

2.3. Корректность алгоритма

Как было сказано выше, алгоритм вычисляет приближенное решение, в итоге будут получены надмножества всех искомых булевых отношений для пар вершин. Результат работы алгоритма будет содержаться в ячейках матрицы T. Корректность алгоритма следует из выполнимости следующих утверждений.

***Лемма* 2.1.** *(А. Охотин) После выполнения процедуры compute(l, m), где m–l – степень двойки, $\{A|a_{i+1}, ..., a_j \in L(G_A) \in T_{i,j}\}$ для всех $l \leq i < j < m$.*

***Теорема* 2.1.** *(Корректность алгоритма для синтаксического анализа ациклических графов). Пусть даны помеченный ациклический граф D и булева грамматика $G = (\Sigma, N, R)$. Тогда для любых вершин i, j и для любого нетерминала $A \in N$, если $(i, j) \in R_A$, то $A \in T_{i,j}$.*

Доказательство. По определению реляционного отношения, если $(i, j) \in R_A$, то существует $i\pi j$, такой, что $l(\pi) \in L(G_A)$. Докажем индукцией по высоте дерева разбора строки $l(\pi)$, что $A \in T_{i,j}$.

Алгоритм 1 Алгоритм для синтаксического анализа ациклических графов

Вход: $D = (V, E)$ — ациклический граф с пронумерованными вершинами, G — булева грамматика.

Выход: матрица T

$n \leftarrow |V|$
$D \leftarrow TopologicalSorting(D)$
$T \leftarrow T[0][0], T[0][1], ..., T[n][n]$
$P \leftarrow P[0][0], P[0][1], ..., P[n][n]$
for $(i, j, a) \in E$ **do**
    $T_{i,j} \leftarrow \{A | A \to a \in R\}$
$compute(0, n-1)$
procedure COMPUTE(l, m)
    if $m - 1 \geq 4$ **then**
        $compute(l, \frac{l+m}{2})$
        $compute(\frac{l+m}{2}, m)$
    $complete(l, \frac{l+m}{2}, \frac{l+m}{2}, m)$
procedure COMPLETE(l, m, l', m')
    if $(l, m, a) \in E$ and $m < l'$ **then**
        $T_{l,l'} \leftarrow f(P_{l,l'})$
    else if $m - 1 > 1$ **then**
        $B \leftarrow (l, \frac{l+m}{2}, \frac{l'+m'}{2}, m')$
        $B' \leftarrow (\frac{l+m}{2}, m, l', \frac{l'+m'}{2})$
        $C \leftarrow (\frac{l+m}{2}, m, l', \frac{l'+m'}{2})$
        $D \leftarrow (l, \frac{l+m}{2}, l', \frac{l'+m'}{2})$
        $D' \leftarrow (\frac{l+m}{2}, m, \frac{l'+m'}{2}, m')$
        $E \leftarrow (l, \frac{l+m}{2}, \frac{l'+m'}{2}, m')$
        $complete(C)$
        $P_D \leftarrow P_D \cup (T_B \times T_C)$
        $complete(D)$
        $P_{D'} \leftarrow P_{D'} \cup (T_C \times T_{B'})$
        $complete(D')$
        $P_E \leftarrow P_E \cup (T_B \times T_{D'})$
        $P_E \leftarrow P_E \cup (T_D \times T_{B'})$
        $complete(E)$

База индукции. Если (i, j) — ребро графа (высота дерева разбора равна 1), то алгоритм корректен, исходя из инициализации матрицы T (строки 5-6 листинга 1).
Индукционный переход. Предположим, утверждение верно для всех деревьев разбора высотой n. Докажем, что теорема верна для деревьев разбора высотой $n + 1$.
Рассмотрим дерево разбора для $l(\pi)$ высотой $n+1$. Так как грамматика в нормальной форме, то у данного дерева будут поддеревья, выводящие подстроки $l(\pi)$ (возможно пересекающиеся). По свойству топологической сортировки для всех индексов k, l этих подстрок (начала и конца соответствующих путей) выполняется неравенство $i \leq k < m \leq j$.
По Лемме 2.1 вызов $compute(i, j)$ посчитает $\{A|a_{i+1}, ..., a_j \in L(G_A)\} \in T_{i,j}$ для всех $i \leq k < l < j$. По индукционному предположению, для всех подпутей $k\pi'm$ пути $i\pi j$ и нетерминалов $A \in T_{k,m}$ верно, что $l(\pi) \in L(G_A)$
$T_{i,j}$ вычисляется как $f(P_{i,j})$. $P_{i,j}$ по алгоритму 1 может быть получено тремя способами, в зависимости от положения подматрицы, в которой оно задано. Пусть $P_{i,j}$ — ячейка матрицы P_Z. В любом из случаев P_Z вычисляется как $P_Z \cup (T_X \times T_Y)$. По определению произведения $T_X \times T_Y$ (3), для каждого пути i, j будут получены все произведения нетерминалов из конъюнктов, выводящих все подстроки $l(\pi)$, такие, что путь $i\pi j$ разбит на две части вершиной k, такой что $i < k < j$. По индукционному предположению и по Лемме 2.1 для всех таких подстрок будут существовать деревья разбора, и высота их не будет превышать n. Тогда исходя из правил грамматики и того, что она в нормальной форме, произведение матриц $T_X \times T_Y$ даст все возможные нетерминалы $A \in T_{i,j}$, такие, что $l(\pi) \in L(G_A)$. А это значит, что утверждение верно для дерева разбора высотой $n + 1$.

Время работы алгоритма аналогично времени работы алгоритма Охотина [24] и составляет $O(|G|BMM(n) \log n)$, где $|G|$ — размер входной булевой грамматики, n — число вершин в графе D, $BMM(n)$ – время умножения булевых матриц размера $n \times n$.

3. Булева схема для синтаксического анализа графов

Булева схема будет построена основе алгоритма Брента-Гольдшлягера-Риттера [6]. Идея алгоритма состоит в построении системы логического вывода и доказательства утверждений вида $A(i, j)$, где A — нетерминал, а i, j — начало и конец пути в D соответственно. В предложенной ниже системе вывода также будут существовать условные утверждения вида "если $D(k, l)$, то $A(i, k :: l, j)$", где $i :: j$ обозначает *дырку*, оставленную в пути из вершины i в вершину j со свойством A для подотрезка этого пути из вершины k в вершину l со свойством D. Дерево разбора для $A(i, j)$ и дерево разбора $A(i, j)$ с дыркой $D(k, l)$ представлены на рис. 4.

3.1. Система вывода

Дан ориентированный граф D с пронумерованными вершинами и контекстно-свободная грамматика в нормальной форме Хомского.
Тогда используется система вывода со следующими правилами:

$\frac{}{A(i,j)}$ (метка ребра $i \xrightarrow{a} j$, если $A \to a \in P$, $i, j \in E$ и помечено символом $a \in \Sigma$)

$\frac{B(i,j)}{\frac{A}{C}(i,j::z)}$ (создание дырки справа, если $A \to BC \in P$ и $z \in V$)

$\frac{C(j,z)}{\frac{A}{B}(i::j,z)}$ (создание дырки слева, если $A \to BC \in P$ и $i \in V$)

$\frac{\frac{A}{D}(i,j::w,z),D(j,w)}{A(i,z)}$ (затыкание дырки)

$\frac{\frac{A}{E}(i,j::w,z),\frac{E}{D}(j,u::v,w)}{\frac{A}{D}(i,u::v,z)}$ (соединение условных утверждений)

3.2. Оценка глубины рекурсии

***Лемма* 3.1.** *Пусть G — грамматика в нормальной форме Хомского. Тогда всякое дерево разбора с ζ листьями содержит вершину (будем называть ее промежуточной), в чьём поддереве более чем $\frac{1}{3}\zeta$ листьев, и менее $\frac{2}{3}\zeta$ листьев.*

Доказательство. Путь строится сверху, каждый раз выбирается наибольшее поддерево. Так как ветвление двоичное, искомая вершина найдется (рис. 5). Данная лемма иллюстрирует тот факт, что дерево разбора можно разбить на три меньших поддерева.

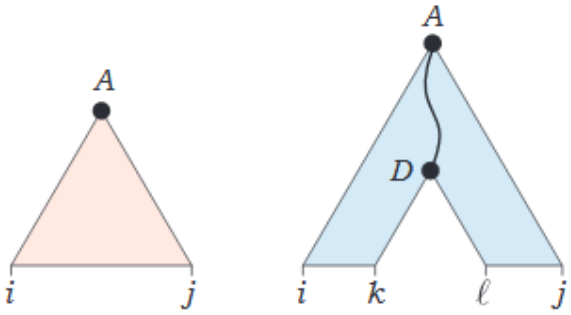

Рис. 4: Дерево разбора для $A(i,j)$ (слева) и дерево разбора $A(i,j)$ с дыркой $D(k,l)$ (справа)

***Лемма* 3.2.** *Пусть ζ — длина пути из вершины i в j. Тогда в системе вывода всякое утверждение $\frac{A}{D}(i,u :: v,j)$ или $A(i,j)$ имеет доказательство высоты не более чем $4\log_{3/2}\zeta$.*

Доказательство. Индукция по ζ.
Если t — путь в графе, то дерево разбора для этого пути будет содержать ζ листьев (каждое ребро помечено терминальным символом). По Лемме 3.1 для дерева разбора $A(i,j)$ найдется промежуточная вершина, назовём её E, а её детей B и C ($E \to BC \in P$). Тогда вывод $A(i,j)$ можно записать следующим образом:

$$\dfrac{\dfrac{B(u,l) \qquad C(l,v)}{E(u,v)} \qquad \frac{A}{E}(i,u :: v,j)}{A(i,j)}$$

По свойству промежуточной вершины, количество листьев в поддеревьях $\frac{A}{E}(i,u :: v,j)$, $B(u,l)$, $C(l,v)$ не превышает $\frac{2}{3}\zeta$. По предположению индукции, для них есть доказательства логарифмической высоты.

Для $\frac{A}{D}(i,u :: v,j)$ рассмотрим два случая: 1) дерево промежуточной вершины включает в себя дырку и 2) дерево промежуточной вершины содержит только листья, относящиеся к пути $i \to u$ (или $v \to j$).
Первый случай. Пусть промежуточная вершина помечена символом E. Так как поддерево этой вершины содержит дырку, то оно разбивает путь $i \to u$ на два пути вида $i \to k$ и $k \to u$, а путь $v \to j$ на $v \to l$ и $l \to j$. Тогда вывод $\frac{A}{E}(i,u :: v,j)$ можно записать так:

$$\dfrac{\frac{A}{E}(i,k :: l,j) \qquad \frac{A}{D}(k,u :: v,l)}{\frac{A}{E}(i,u :: v,j)}$$

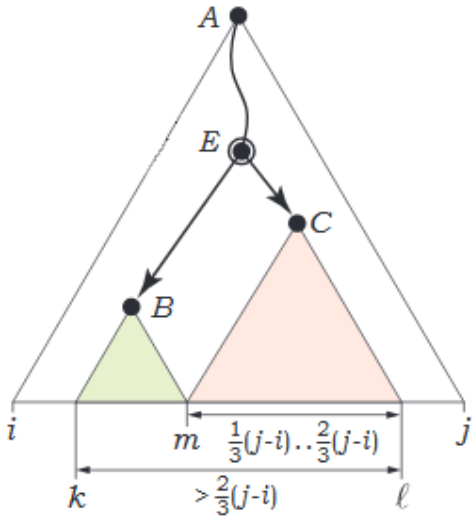

Рис. 5: Промежуточная вершина E и два её поддерева

Второй случай. Пусть дерево промежуточной вершины содержит только листья, относящиеся к пути $i \to u$. Рассмотрим наибольшее поддерево, включающее данное, и содержащее только листья, относящиеся к пути $i \to u$. Пусть это поддерево помечено нетерминалом B и выводит путь $k \to l$, являющийся подпутем $i \to u$. Пусть E - родитель B, а C - второй потомок E ($E \to BC \in P$). Тогда будет разбивать путь $v \to j$ на $v \to m$ и $m \to j$. Тогда вывод для $\frac{A}{E}(i, u :: v, j)$ выглядит так:

$$\dfrac{\dfrac{\dfrac{}{B(k,l)}}{\frac{E}{C}(k, l :: m)} \quad \frac{C}{D}(l, u :: v, m)}{\frac{E}{D}(k, u :: v, m)} \quad \frac{A}{E}(i, k :: m, j)}{\frac{A}{D}(i, u :: v, j)}$$

Так как $B(k, l)$ содержит не менее чем $\frac{1}{3}\zeta$ листьев, то утверждения $\frac{A}{E}(i, k :: m, j)$ и $\frac{C}{D}(l, u :: v, m)$ содержат менее $\frac{2}{3}\zeta$ листьев каждое, поэтому по предположению индукции они имеют доказательство логарифмической высоты. Для утверждений вида $B(k, l)$ уже доказано существование доказательства логарифмической высоты. Получаем, что дерево $\frac{A}{D}(i, u :: v, j)$ с ζ листьями выводится за 4 шага из поддеревьев не более чем с $\frac{2}{3}\zeta$ листьями.

***Лемма* 3.3.** *Пусть ζ — максимальная длина пути, являющегося ответом на задачу поиска путей с контекстно-свободными ограничениями. Тогда глубина булевой схемы для решения данной задачи составляет $O(\log^2 \zeta)$.*

Доказательство. На стеке на каждом уровне рекурсии осуществляется не более $\log \zeta$ операций, глубина рекурсии по Лемме 3.2 — $O(\log \zeta)$.

***Следствие* 3.4.** *Пусть дан ациклический граф D с $|V| = n$ и контекстно-свободная грамматика G. Тогда глубина схемы для синтаксического анализа ациклического графа — $O(\log^2 n)$.*

Доказательство. Произведем топологическую сортировку графа D. Так как максимальная длина пути в таком графе не превысит n, то ζ также меньше n. Отсюда получаем искомую оценку.

3.3. Описание схемы

Вход: Элементы вида (a, i, j), где i, j — смежные вершины графа, $a \in \Sigma$ — помечено ли ребро из i в j перминалом a. Количество входных элементов — $|\Sigma| \cdot n^2$

Элементы:

- Для всех вершин i, j, где $1 \leq i, j \leq n$, есть элемент $x_{A,i,j}$, в котором вычисляется значение $A(i, j)$, то есть принадлежит ли слово, составленное из меток пути $i \to ... \to j$ языку $L_G(A)$.
- Элемент $y_{A,i,j,D,k,l}$, где $A, D \in N, 1 \leq i, j, k, l \leq n$. Этот элемент определяет, существует ли дерево разбора пути $i \to ... \to j$ из A, с дыркой вместо поддерева $k \to ... \to l$ из D, так что в нём вычисляется значение 1 тогда и только тогда, когда верно условное утверждение $\frac{A}{D}(i \to ... \to k :: l \to ... \to j)$.

Выход: Элементы вида (A, i, j) — существует ли поддерево разбора с нетерминалом A в корне для строки, полученной конкатенацией терминальных символов ребер пути из вершины i в j. Таких элементов $|N| \cdot n^2$.

Данные вычисления можно адаптировать для модели параллельных вычислений SIMDAG.

3.4. Описание алгоритма для модели SIMDAG

Параллельный алгоритм для решения задачи поиска путей с контекстно-свободными ограничениями для модели параллельных вычислений SIMDAG аналогичен модифицированному алгоритму Брента-Гольдшлягера-Риттера [6]. Модификации включают в себя:

- снятие ограничения на порядок индексов (например, требование для $A(i, j)$ вида $i \leq j$) — данные модификации носят формальный характер, так как не влияют на порядок и состав вычислений в оригинальном алгоритме
- добавление выходных элементов — для каждого элемента $x_{A,i,j}$ добавлен аналогичный выходной, так как для решения задачи контекстно свободной-достижимости

необходимо ответить на вопрос о достижимости всех возможных видов путей в графе

- увеличение количества итераций до $4 \log_{3/2} t$, где t — максимальная длина строки, являющейся ответом на задачу.

Ниже приведен псевдокод модифицированного алгоритма:

Алгоритм 2 Алгоритм Брента-Гольдшлягера-Риттера для решения задачи поиска путей с контекстно-свободными ограничениями

1: $P(x_{A,i,j}) \leftarrow P[0], P[1], ..., P[Nn^2 - 1]$
2: $Q(y_{A,i,j,D,k,l}) \leftarrow Q[0][0], Q[0][1], ..., Q[Nn^2 - 1][Nn^2 - 1]$
3: **if** $(i, a, j) \in E$ and $A \to a$ **then**
4: $\quad P(x_{A,i,j}) \leftarrow$ true
5: **for** $t = 1$ to $4 \log_{3/2} t$ **do**
6: $\quad U(y_{A,i,j,D,k,l}) \leftarrow Q(y_{A,i,j,D,k,l})$
7: $\quad$ **if** $(C \to DE) \in P$ and $((Q(y_{D,p,q,B,x,y})$ and $P(x_{E,q,r}))$ or $(P(x_{D,p,q})$ and $(Q(y_{E,q,r,B,x,y})))$ **then**
8: $\quad\quad U(y_{C,p,r,B,x,y}) \leftarrow$ true
9: $\quad$ **if** $U(y_{A,i,j,C,p,r})$ and $U(y_{C,p,r,B,x,y})$ **then**
10: $\quad\quad Q(y_{A,i,j,B,x,y}) \leftarrow$ true
11: $\quad V(x_{A,i,j}) \leftarrow P(x_{A,i,j})$
12: $\quad$ **if** $(C \to DE) \in P$ and $P(x_{D,p,q})$ and $P(x_{E,q,r})$ **then**
13: $\quad\quad V(x_{C,p,r}) \leftarrow$ true
14: $\quad$ **if** $Q(y_{A,i,j,C,p,r})$ and $V(x_{C,p,r})$ **then**
15: $\quad\quad P(x_{C,p,r}) \leftarrow$ true

Оценка времени работы и количества процессоров

Операцией, которая задействует наибольшее количество элементов в исходном алгоритме, является операция соединения условных утверждений: $U(y_{A,i,j,C,p,r}) \wedge U(y_{C,p,r,B,x,y})$. Она требует $N^2 n^4 \times Nn^2 = N^3 n^6$ элементов. В модифицированной версии общее количество элементов не меняется, за исключением добавления дополнительных выходных элементов, число которых Nn^2. Поэтому количество элементов в схеме $O(n^6)$, если считать число нетерминалов константой.

Из Леммы 3.3 и построения алгоритма следует, что его время работы равно $O(\log^2 t)$ на unit-cost SIMDAG и $O(\log t)$ на log-cost SIMDAG.

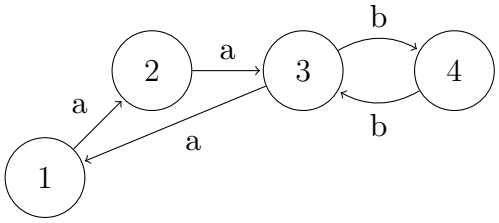

Рис. 6: Помеченный ориентированный граф

4. Оценка глубины схемы для различных классов контекстно-свободных грамматик

В частях 2.3 - 2.5 будут анализироваться деревья разбора для грамматики в разных нормальных формах: нормальная форма Хомского (2.3) и квадратичная нормальная форма Грейбах (2.5). Перевод в НФХ увеличивает высоту дерева в константу раз, то же самое увеличение дает перевод в форму Грейбах [19]. Таким образом, конкретная форма грамматики асимптотически не повлияет на анализируемые параметры.

4.1. Минимальная строка

В предыдущем разделе было показано, что высота схемы зависит от минимальной длины строки ζ, являющейся ответом на задачу. Определим её формально следующим образом.

Пусть дана контекстно-свободная грамматика G, где $A \in N$ и помеченный ориентированный граф D, где $m, n \in V$, такие что $\mathcal{L} = \mathcal{L}(G; A) \cap \mathcal{L}(D; m, n)$. Тогда $\zeta = \max\limits_{\forall(m,n)} \min ||\mathcal{L}||$.

Например, для графа на рис. 6 и контекстно-свободной грамматики $G = (\{S\}, \{a, b\}, \{S \to aSb, S \to \varepsilon\})$ $\zeta = 12$ (путь из вершины с номером 3 в вершину 3, соответствующая последовательность — "aaaaaabbbbbb").

В работе [16] даны следующие верхняя и нижняя оценки на ζ:

- Нижняя оценка: $\zeta \geq \frac{2^{|N|}n^2}{64}$.
- Верхняя оценка: $\zeta \leq 2^{|N|n^2}$,

где n — число вершин в графе, $|N|$ — число нетерминалов в контекстно-свободной грамматике.

4.2. Рациональный индекс

В связи с тем, что для схемы фиксирована грамматика, а граф подаётся на вход, интересно рассмотреть взаимосвязь ζ и числа вершин в графе при фиксированной кс-

грамматике. Такую связь описывает специальный показатель сложности контекстно-свободных языков — *рациональный индекс*. Ниже дано его определение.

Пусть фиксирован некоторый контекстно-свободный язык L. Пусть Rat_n — семейство регулярных языков, распознаваемых конечным автоматом с не более чем n состояниями. Тогда *рациональный индекс* $p_L(n)$ языка L определяется как функция вида $\mathbb{N} \to \mathbb{N}$:

$$p_L(n) = \max_{K \in Rat_n | K \cap L \neq \emptyset} \min_{w \in K \cap L} |w| \tag{5}$$

Нетрудно заметить, что рациональный индекс описывает поведение величины ζ для фиксированного кс-языка в зависимости от увеличения количества вершин в графе (действительно, каждому графу можно однозначно сопоставить конечный автомат, а конечному автомату — регулярный язык). Поэтому, зная величину рационального индекса, можно получить верхнюю и нижнюю оценки высоты схемы для распознавания данного языка.

На текущий момент известны верхние оценки для рационального индекса следующих языков:

- Язык Дика на одном типе скобок $D_1'^* = (\{S\}, \{a, b\}, \{S \to aSbS, S \to \varepsilon\})$ — $p_L(n) = O(n^2)$ [10].
- Языки Грейбах H — $p_L(n) = \Theta(n^\lambda)$, где λ — алгебраическое число больше 1 [28].
- Произвольный контекстно-свободный язык — $p_L(n) = \Theta(2^{\frac{n^2}{\ln n}})$ [27].

Исходя из Леммы 3.3, получаем:

***Следствие* 4.1.** *Пусть дан граф D с $|V| = n$ и контекстно-свободная грамматика G. Тогда в общем случае глубина схемы для синтаксического анализа графа будет полиномиальной от входа $t = \Sigma n^2$ — $O(t^2)$ для константного Σ, в случае если $L(G) = D_1'^*$ или $L(G) = H$ полилогарифмической — $O(\log^2 n)$.*

4.3. Оценка глубины схемы для линейных грамматик

Линейная грамматика — контекстно-свободная грамматика, которая имеет не более одного нетерминала в правой части каждого правила.
Пусть T — множество терминалов. Линейная грамматика $G = (N, T, \Sigma, P)$ находится в *нормальной форме Хомского* тогда, когда любое её правило имеет одну из следующих форм: $A \to aB$, $A \to Ba$, $A \to a$, где $B \in N$ и $a \in T$.

***Лемма* 4.2.** *Пусть дана контекстно-свободная грамматика G, где $A \in N$ и помеченный ориентированный граф D, где $m, n \in V$, такие что $\mathcal{L} = \mathcal{L}(G; A) \cap \mathcal{L}(D; m, n)$. Пусть $l = \min ||\mathcal{L}||$. Тогда высота дерева разбора строки l не превышает $|N|V^2$ для любых m, n.*

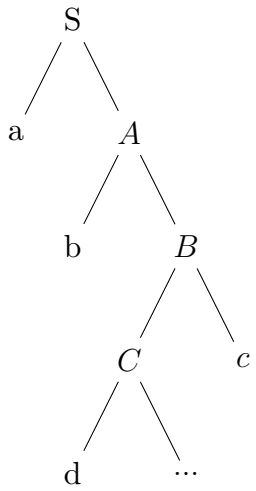

Рис. 7: “Худшее“ дерево разбора для input-driven грамматики

Доказательство. Пусть у дерева разбора l высота больше, чем $|N|V^2$. Так как l минимальная, то все её подстроки минимальны (в противном случае мы могли бы заменить большую на меньшую). Уникальных троек вида (A, i, j) всего $|N|V^2$, тогда по принципу Дирихле в дереве разбора для l будет как минимум одно поддерево T, соответствующее какой-либо из троек и содержащее поддерево T', соответствующее такой же тройке, но выводящее строку длиной меньше. Тогда заменив T на T' получим строку длиной меньше, чем l. Но по условию l — минимальная, получаем противоречие.

***Теорема* 4.1.** *Глубина схемы синтаксического разбора графа D на n вершинах для случая линейной грамматики G равна $O(\log^2 n)$.*

Доказательство. Оценим величину показателя ζ для линейной грамматики. Рассмотрим дерево разбора для строки длины ζ, по Лемме 4.2 его высота не превосходит $|N|n^2$, оно изображено на рис. 7. Будем спускаться от корня дерева вниз, считая листья. Так как линейная грамматика в нормальной форме Хомского, то спуск на один уровень добавит только один лист к общему количеству листьев. На самом последнем уровне по правилу $A \to a$ дерево закончится. Отсюда получаем, что количество листьев равно высоте дерева, а именно не превосходит $|N|n^2$. Тогда $\zeta \leq |N|n^2$, а значит высота схемы в случае линейной грамматики равна $\log^2(|N|n^2) = O(\log^2 n)$ при константном числе нетерминалов.

4.4. Оценка глубины схемы для input-driven грамматик

Input-driven грамматика $G = (\Sigma, N, S, P)$ определена следующим образом [1]:

- $\Sigma = \Sigma_{+1} \cup \Sigma_0 \cup \Sigma_{-1}$ — алфавит, разделенный на три непересекающихся класса.
- N — множество нетерминальных символов.

- S — стартовый нетерминал, $S \in N$.
- P — набор правил, каждое из которых находится в следующей форме: либо $A \to < B > C$, либо $A \to aC$, либо $A \to \varepsilon$, где $A, B, C \in N$, $< \in \Sigma_{+1}$, $> \in \Sigma_{-1}$, $a \in \Sigma$.

Примером input-driven языка является язык $L = \{a^n b^n | n \geq 2\} \cup \{a^n c^n | n \geq 2\}$.
Input-driven язык и распознаются специальным видом автоматов — *автоматами с магазинной памятью, управляемыми входом (IDPDA)*. Дадим формальное определение таких автоматов [26]:
Детерминированный автомат с магазинной памятью, управляемый входом (DIDPDA) определён как семёрка $A = (\Sigma, Q, \Gamma, q_0, \bot, [\delta_a], F)$, где:

- Q — конечное множество состояний автомата с начальным состоянием $q_0 \in Q$ и множеством конечных состояний $F \subseteq Q$.
- Γ — конечный стековый алфавит, символ $\bot \notin \Gamma$ означает пустой стек.
- Функция перехода для открывающей скобки $< \in \Sigma_{+1}$ — частичная функция $\delta_< : Q \to Q \times \Gamma$, которая для текущего состояний показывает следующее состояние и символ, добавляемый в стек.
- Функция перехода для закрывающей скобки $> \in \Sigma_{-1}$ — частичная функция $\delta_> : Q \times (\Gamma \cup \{\bot\}) \to Q$, которая для текущего состояний показывает следующее состояние, подразумевая, что указанный символ вынут из стека или стек пустой.
- Для нейтрального символа $c \in \Sigma_0$, изменение состояние описывается частичной функцией $\delta_c : Q \to Q$.

Конфигурация A — тройка (q, w, x), где $q \in Q$ — состояние, $w \in \Sigma^*$ — оставшийся вход и $x \in \Gamma^*$ — содержимое стека. Начальная конфигурация при входной строке $w_0 \in \Sigma^*$— (q_0, w_0, ε). Для каждой конфигурации с как минимум одним оставшимся символом следующая конфигурация определяется единственным образом согласно следующим правилам:

- Для каждой открывающей скобки $< \in \Sigma_{+1}$ $(q, < w, x) \vdash_A (q', w, \gamma x)$, где $\delta_<(q) = (q', \gamma)$.
- Для каждой открывающей скобки $> \in \Sigma_{-1}$ $(q, > w, \gamma x) \vdash_A (\delta > (q, \gamma), w, x)$, а в случае пустоты стека $(q, > w, \varepsilon) \vdash_A (\delta_>(q, \bot), w, \varepsilon)$.
- Для нейтрального символа $c \in \Sigma_0$ $(q, cw, x) \vdash_A (\delta_0(q), w, x)$.

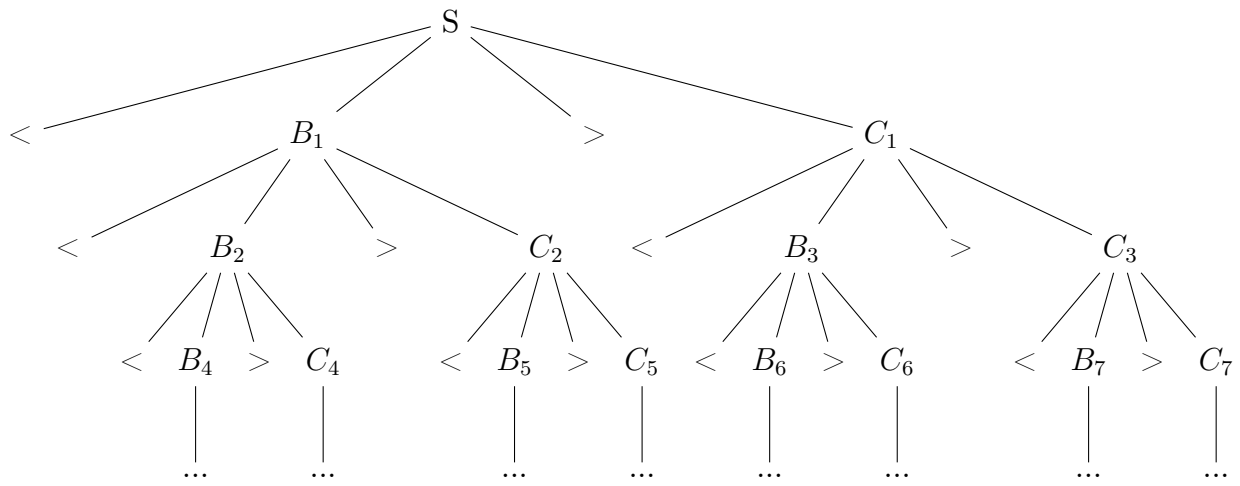

Рис. 8: “Худшее“ дерево разбора для input-driven грамматики

Конфигурация (q, ε, u) является принимающей, когда $q \in F$, независимо от содержимого стека. Язык $L(A)$, распознаваемый данным автоматом — набор всех строк $w \in \Sigma^*$, на которых вычисленная конфигурация (q, ε, u) является принимающей.

Недетерминированный автомат с магазинной памятью, управляемый входом (NIDPDA) определен схожим образом, за исключением возможности наличия нескольких начальных состояний и тем, что для каждого входного символа a функция перехода δ_a может осуществить переход в несколько состояний.

Найдем верхнюю границу для ζ для input-driven грамматик.
Для этого построим “худшее“ дерево разбора и оценим количество листьев в нём. “Худшим“ назовём дерево с максимально возможным количеством листьев. Будем строить дерево сверху вниз. Для того, чтобы выводимая строка была максимальной, задействуем правила вида $A \to < B > C$ — это правило даёт наибольшее количество листьев при применении. Пример построения показан на рис. 8.
По Лемме 4.2 в случае синтаксического анализа графа на n вершинах высота данного дерева не превысит $|N|n^2$. Нетрудно заметить, что на каждом уровне дерева прибавляется в два раза больше листьев. По формуле суммы геометрической прогрессии получаем длину строки для данного дерева разбора:

$$\zeta \leq 2^{|N|n^2-1} - 2 \tag{6}$$

***Следствие* 4.3.** *Глубина схемы синтаксического разбора графа D вершинах для случая input-driven грамматики G равна $O(|N|n^2)$ для входа длиной n.*

4.5. Оценка глубины схемы для LL(k) грамматик

Пусть дана контекстно-свободная грамматика $G = (N, \Sigma, P, S)$. Рассмотрим два произвольных левосторонних вывода слова w в этой грамматике:

- $S \Rightarrow^* pA\beta \Rightarrow p\alpha\beta \Rightarrow^* py\eta$
- $S \Rightarrow^* pA\beta \Rightarrow p\alpha'\beta \Rightarrow^* py\xi$

где p и y — цепочки из терминалов, уже разобранная часть слова w, A — нетерминал грамматики, в которой есть правила $A \to \alpha$ и $A \to \alpha'$, причем $\alpha, \alpha', \beta, \eta, \xi$ — последовательности из терминалов и нетерминалов. Если из выполнения условий, что $|y| = k$ или $|y| < k$, $\eta = \xi = \varepsilon$, следует равенство $\alpha = \alpha'$, то G называется *LL(k)-грамматикой* [33].

Практическое преимущество LL-грамматик состоит в том, что они позволяют *синтаксический анализ сверху-вниз* — анализ, при котором анализатор исходит из предположения,что входная строка синтаксически верна, и, читая строку слева направо, обходит предполагаемое дерево разбора сверху вниз.

Оценим влияние величины параметра k на структуру дерева разбора. По определению LL-грамматики, для каждого нетерминала и входной строки терминалов длиной $\leq k$ должно существовать единственное правило вывода, которое однозначно определяется этой строкой и текущим нетерминалом при спуске сверху-вниз. Всего правил для одного нетерминала не более $\frac{\Sigma(\Sigma^{k-1}-1)}{\Sigma-1}$. С точки зрения разбора графа, каждому нетерминалу должен соответствовать путь из i в j. Тогда для каждого нетерминала должно существовать не менее одного уникального под пути длиной $\leq k$. Таких подпутей может быть максимум $\frac{\Sigma(\Sigma^{k-1}-1)}{\Sigma-1}$. Нас интересуют пути из i в j для каждого нетерминала, таких путей n^2. У n^2 путей должно быть как минимум n начальных подпутей. Получаем $n \leq \frac{\Sigma(\Sigma^{k-1}-1)}{\Sigma-1}$ для каждого нетерминала. Пусть $k = 1$, тогда $n < \Sigma$, такую грамматику и граф можно подобрать, тогда в его дереве разбора будут существовать все тройки. Для $k > 1$ подбор ещё легче сделать, так как правая часть неравенства растёт со скоростью геометрической прогрессии.

Определение накладывает большие ограничения на грамматику. В частности, нельзя использовать левую рекурсию: если она есть, то анализатор зациклится, будет наращивать в стеке периодическую последовательность. Нормальный вид Грейбах, в котором все правила имеют вид $A \to a\alpha$, где $a \in \Sigma$ и $\alpha \in (\Sigma \cup N)^*$, не только исключает возможность левой рекурсии, но и вообще удобен для обработки строки слева направо. Пусть грамматика находится в квадратичной нормальной форме Грейбах. Грамматика находится в *квадратичной нормальной форме Грейбах* тогда и только тогда, когда она в нормальной форме Грейбах и в правой части каждого правила не более двух нетерминалов. Розенкранц в работе [32] доказал, что любая контекстно-свободная грамматика может быть представлена в квадратичной нормальной форме Грейбах. Поэтому построим “худшее дерево“ для LL-грамматики в этой форме (рис. 9). Предположим, что у нас соблюдаются выше описанные условия на соотношение Σ и n.

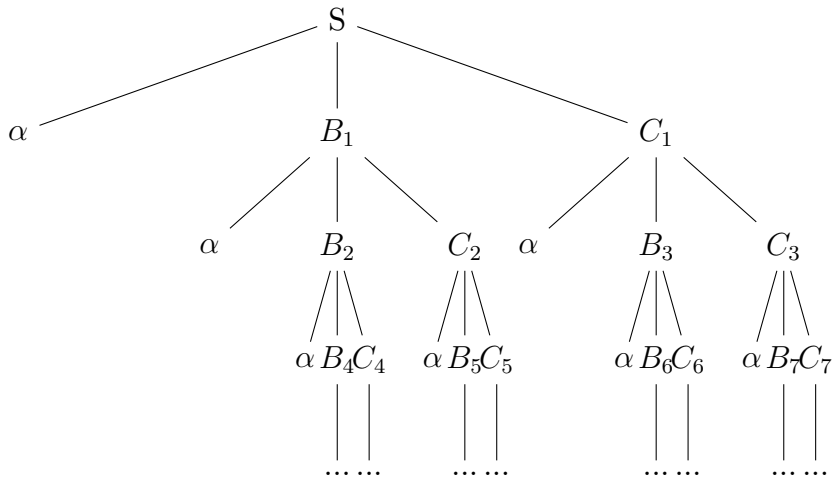

Рис. 9: “Худшее“ дерево разбора для грамматики в квадратичной нормальной форме Грейбах

Так как высота дерева не превышает $|N|n^2$, а количество листьев на каждом уровне растет со скоростью геометрической прогрессии со знаменателем 2, получаем:

$$\zeta \leq 2^{|N|n^2-1} - 1 \tag{7}$$

***Следствие* 4.4.** *Глубина схемы синтаксического разбора графа D для случая LL(k) грамматики G равна $O(|N|n^2)$ для входа длиной n.*

Заключение

В рамках данной работы был предложен алгоритм синтаксического анализа ациклических графов. Благодаря топологической сортировке, которая задает порядок на вершинах графа, становится возможным использовать классические алгоритмы синтаксического анализа, использующие стратегию динамического программирования. Таким образом, на ациклических графах задачу можно решить за время умножения булевых матриц, как и обычную задачу синтаксического анализа на строках. Более того, предложенный алгоритм работает не только с контекстно-свободными грамматиками, но и с их расширением — булевыми грамматиками, что позволяет формировать более сложные запросы к графам.

Также была построена булева схема для решения задачи и дан адгоритм, обобщающий данную схему для модели SIMDAG. В общем случае, задача не может быть эффективно решена с помощью параллельной схемы, так как глубина схемы полиномиальна ($O(|N|n^2)$) от входа размером n для произвольной контекстно-свободной грамматики. Но было показано, что глубина схемы зависит от максимальной длины строки, являющейся ответом на задачу. Данный показатель может быть полиномиальным от количества вершин в графе D, что даст эффективную схему логарифмической глубины. В работе показано, что к таким классам относятся линейный грамматики и языки Дика на одном типе скобок. Также данный показатель линеен от количества вершин в графе, в случае, если входной граф является ациклическим, поэтому для таких графов булева схема будет работать эффективно и иметь также полилогарифмическую глубину. К классам грамматик с максимальной строкой экспоненциального размера относятся input-driven грамматики и LL-грамматики, поэтому глубина схемы для них остается полиномиальной от размера входа.

Список литературы

[1] Alur Rajeev, Madhusudan P. Visibly Pushdown Languages // Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing. — STOC '04. — New York, NY, USA : ACM, 2004. — P. 202–211.

[2] Azimov Rustam, Grigorev Semyon. Graph Parsing by Matrix Multiplication // CoRR. — 2017. — Vol. abs/1707.01007. — 1707.01007.

[3] Azimov Rustam, Grigorev Semyon. Path querying using conjunctive grammars // Proceedings of the Institute for System Programming of the RAS. — 2018. — 01. — Vol. 30. — P. 149–166.

[4] Barceló Baeza Pablo. Querying graph databases // Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. — 2013. — 06.

[5] Bradford Phillip G. Efficient Exact Paths For Dyck and semi-Dyck Labeled Path Reachability // CoRR. — 2018. — Vol. abs/1802.05239. — 1802.05239.

[6] Brent Richard, M. GOLDSCHLAGER LESLIE. A PARALLEL ALGORITHM FOR CONTEXT-FREE PARSING. — 1983. — 01.

[7] Chaudhary Anoop, FAISAL ABDUL. Role of graph databases in social networks. — 2016. — 06.

[8] Circuits and Expressions over Finite Semirings / Moses Ganardi, Danny Hucke, Daniel König, Markus Lohrey // ACM Transactions on Computation Theory. — 2018. — 08. — Vol. 10. — P. 1–30.

[9] Context-Free Path Queries on RDF Graphs / Xiaowang Zhang, Zhiyong Feng, Xin Wang, Guozheng Rao // CoRR. — 2015. — Vol. abs/1506.00743. — 1506.00743.

[10] Deleage Jean-Luc, Pierre Laurent. The rational index of the Dyck language D1∗ // Theoretical Computer Science. — 1986. — Vol. 47. — P. 335 – 343.

[11] Dymond Patrick W. Input-driven languages are in log n depth // Information Processing Letters. — 1988. — Vol. 26, no. 5. — P. 247 – 250.

[12] Earley Jay. An Efficient Context-free Parsing Algorithm // Commun. ACM. — 1970. — Feb. — Vol. 13, no. 2. — P. 94–102.

[13] Greenlaw Raymond, Hoover H. James, Ruzzo Walter L. Limits to Parallel Computation: P-completeness Theory. — New York, NY, USA : Oxford University Press, Inc., 1995. — ISBN: 0-19-508591-4.

[14] H. Ibarra Oscar, Jiang Tao, Ravikumar Bala. Some Subclasses of Context-Free Languages In NC1. // Inf. Process. Lett. — 1988. — 10. — Vol. 29. — P. 111–117.

[15] Hellings Jelle. Conjunctive Context-Free Path Queries // ICDT. — 2014.

[16] Hellings Jelle. Path Results for Context-free Grammar Queries on Graphs // CoRR. — 2015. — Vol. abs/1502.02242. — 1502.02242.

[17] Holzer Markus, Lange Klaus Jörn. On the complexities of linear LL(1) and LR(1) grammars // Fundamentals of Computation Theory / Ed. by Zoltán Ésik. — Berlin, Heidelberg : Springer Berlin Heidelberg, 1993. — P. 299–308.

[18] Kasami Tadao. AN EFFICIENT RECOGNITION AND SYNTAX ANALYSIS ALGORITHM FOR CONTEXT-FREE LANGUAGES. — 1965. — 07. — P. 40.

[19] Kelemenová Alica. Complexity of normal form grammars // Theoretical Computer Science. — 1983. — Vol. 28, no. 3. — P. 299 – 314.

[20] Komarath Balagopal, Sarma Jayalal, Sunil K. S. On the Complexity of L-reachability // CoRR. — 2017. — Vol. abs/1701.03255. — 1701.03255.

[21] Koschmieder André, Leser Ulf. Regular Path Queries on Large Graphs // Proceedings of the 24th International Conference on Scientific and Statistical Database Management. — SSDBM'12. — Berlin, Heidelberg : Springer-Verlag, 2012. — P. 177–194.

[22] Mendelzon Alberto O., Wood Peter T. Finding Regular Simple Paths in Graph Databases // SIAM J. Comput. — 1995. — Dec. — Vol. 24, no. 6. — P. 1235–1258.

[23] Okhotin Alexander. Boolean grammars // Information and Computation. — 2004. — Vol. 194, no. 1. — P. 19 – 48.

[24] Okhotin Alexander. Parsing by matrix multiplication generalized to Boolean grammars // Theoretical Computer Science. — 2014. — Vol. 516. — P. 101 – 120.

[25] Okhotin Alexander. A Tale of Conjunctive Grammars: 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings. — 2018. — 01. — P. 36–59. — ISBN: 978-3-319-98653-1.

[26] Okhotin Alexander, Salomaa Kai. Complexity of Input-driven Pushdown Automata // SIGACT News. — 2014. — Jun. — Vol. 45, no. 2. — P. 47–67.

[27] Pierre Laurent. Rational indexes of generators of the cone of context-free languages // Theoretical Computer Science. — 1992. — Vol. 95, no. 2. — P. 279 – 305.

[28] Pierre Laurent, Farinone Jean-Marc. Context-free languages with rational index in $\Theta(n^\gamma)$ for algebraic numbers γ // RAIRO - Theoretical Informatics and Applications - Informatique Théorique et Applications. — 1990. — Vol. 24, no. 3. — P. 275–322.

[29] Quantifying variances in comparative RNA secondary structure prediction / James WJ Anderson, Ádám Novák, Zsuzsanna Sükösd et al. // BMC Bioinformatics. — 2013. — May. — Vol. 14, no. 1. — P. 149.

[30] Reps Thomas. Program Analysis via Graph Reachability // Proceedings of the 1997 International Symposium on Logic Programming. — ILPS '97. — Cambridge, MA, USA : MIT Press, 1997. — P. 5–19.

[31] Reutter Juan L., Romero Miguel, Vardi Moshe Y. Regular Queries on Graph Databases // Theor. Comp. Sys. — 2017. — Jul. — Vol. 61, no. 1. — P. 31–83.

[32] Rosenkrantz Daniel. Matrix Equations and Normal Forms for Context-Free Grammars // J. ACM. — 1967. — 07. — Vol. 14. — P. 501–507.

[33] Rosenkrantz D.J., Stearns R.E. Properties of deterministic top-down grammars // Information and Control. — 1970. — Vol. 17, no. 3. — P. 226 – 256.

[34] Rubtsov Alexander A., Vyalyi Mikhail N. Regular realizability problems and context-free languages // CoRR. — 2015. — Vol. abs/1503.00295. — 1503.00295.

[35] Ruzzo Walter L. On uniform circuit complexity // Journal of Computer and System Sciences. — 1981. — Vol. 22, no. 3. — P. 365 – 383.

[36] Rytter Wojciech. On the recognition of context-free languages // Computation Theory / Ed. by Andrzej Skowron. — Berlin, Heidelberg : Springer Berlin Heidelberg, 1985. — P. 318–325.

[37] Swernofsky Joseph, Wehar Michael. On the Complexity of Intersecting Regular, Context-Free, and Tree Languages // ICALP. — 2015.

[38] Tarjan R. Depth-first search and linear graph algorithms // 12th Annual Symposium on Switching and Automata Theory (swat 1971). — 1971. — Oct. — P. 114–121.

[39] Valiant Leslie G. General context-free recognition in less than cubic time // Journal of Computer and System Sciences. — 1975. — Vol. 10, no. 2. — P. 308 – 315.

[40] Younger Daniel H. Recognition and parsing of context-free languages in time n3 // Information and Control. — 1967. — Vol. 10, no. 2. — P. 189 – 208.

[41] Yuan Hao, Eugster Patrick. An Efficient Algorithm for Solving the Dyck-CFL Reachability Problem on Trees // Programming Languages and Systems / Ed. by Giuseppe Castagna. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2009. — P. 175–189.

[42] Zhang Qirun, Su Zhendong. Context-sensitive Data-dependence Analysis via Linear Conjunctive Language Reachability // SIGPLAN Not. — 2017. — Jan. — Vol. 52, no. 1. — P. 344–358.

[43] Łukasz Warchał. Using Neo4j graph database in social network analysis. — 2012.

[44] Компиляторы: принципы, технологии и инструментарий / Д. Ульман, Р. Сети, М. Лам, А. Ахо. — ЛитРес, 2018. — ISBN: 9785041399252.

[45] Хопкрофт Д. Мотвани Р. Ульман Д. Введение в теорию автоматов, языков и вычислений, 2-е издание. — Издательский дом Вильямс. — ISBN: 9785845902610.