

Оглавление

Введение	2
1. Обзор литературы	5
1.1. Решения задачи в общем случае	5
1.2. Нижние оценки	6
1.3. Решения задачи в частных случаях	7
1.4. Выводы и результаты по главе	10
2. Алгоритм, основанный на пересечении языков	11
2.1. Сведение к задаче достижимости для РКА	11
2.2. Решение задачи достижимости для РКА	14
2.3. Получение решений для частных случаев	16
2.4. Выводы и результаты по главе	17
3. Результаты (1)	18
3.1. Алгоритм, основанный на неориентированном транзитивном замыкании	18
3.2. Корректность для неориентированных графов и некоторых классов грам- матик	20
3.3. Корректность для двунаправленных графов и языка Дика	20
3.4. Выводы и результаты по главе	23
4. Результаты (2)	24
4.1. Алгоритм, основанный на неинкрементальном транзитивном замыкании	24
4.2. Алгоритм для языка Дика на одном типе скобок	25
4.2.1. Грамматика для языка \mathcal{D}_1	26
4.2.2. Корректность алгоритма	27
4.2.3. Время работы	28
4.3. Выводы и результаты по главе	29
5. Результаты (3)	30
5.1. Алгоритм для смешанного языка Дика	30
5.2. Выводы и результаты по главе	32
Заключение	33
Список литературы	34

Введение

Актуальность

Графовые модели данных широко используются в различных областях науки, например, в биоинформатике [47], анализе социальных сетей [60, 11], графовых базах данных [33, 54] и разных видах статического анализа [41].

Одной из важных задач в анализе графовых моделей данных является поиск путей с заданными ограничениями. Одним из способов задавать такие ограничения являются формальные языки: если на рёбрах графа написаны метки из фиксированного алфавита, то можно искать пути, конкатенация меток на которых принадлежит фиксированному языку [7]. Например, хорошо изучена задача поиска путей с ограничениями, заданными регулярными языками [35]. В этой же работе мы остановимся на классе контекстно-свободных языков, так как они позволяют решать более широкий класс задач.

Задача поиска путей с контекстно-свободными ограничениями, или, сокращённо, CFPQ¹ была впервые сформулирована в терминах запросов к графовым базам данных [54], но нашла применение и в прочих отраслях, использующих графовые модели. За более чем 30 лет, прошедших с тех пор, было предложено множество разных алгоритмов для её решения [34, 23, 44], в большинстве своём основанных на различных методах синтаксического анализа.

К сожалению, все существующие решения задачи в общем случае недостаточно эффективны для использования на практике [19]. Более того, существует условная нижняя оценка [22], согласно которой, скорее всего, достаточно быстрых решений задачи CFPQ в общем случае и не существует.

Всё вышесказанное приводит к тому, что имеет смысл разрабатывать алгоритмы для частных случаев задачи, имеющие время работы лучше, чем общее решение. Для некоторых из этих случаев уже были построены подобные решения, например для языков Дика на двудirectionalных графах [58, 10]. Проблема существующих решений в том, что они слишком Ad hoc, то есть построены специально под конкретный частный случай, а потому применённые в них подходы и идеи невозможно переиспользовать при построении решений для других частных случаев.

Данная работа нацелена на создание единого подхода к построению решений задачи для CFPQ и применение этого подхода для разработки на его основе новых решений для некоторых частных случаев задачи.

¹Context-Free Path Querying

Постановка задачи

Определение 0.1. *Ориентированный помеченный граф (или граф с метками) — это тройка $G = \langle V, E, \Sigma \rangle$, где V — множество вершин, Σ — множество меток, $E \subseteq V \times V \times \Sigma$ — множество рёбер.*

Неформально, это мультиграф, каждому ребру которого сопоставлена метка из алфавита Σ .

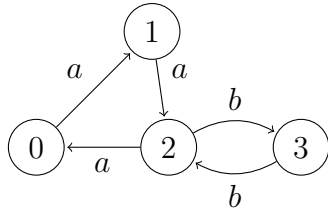


Рис. 1: Помеченный граф с $\Sigma = \{a, b\}$

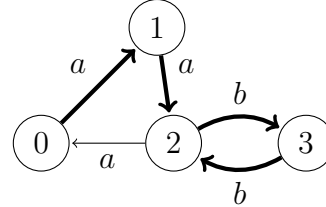


Рис. 2: На пути $v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_2$ читается слово $aabb$

Определение 0.2. Будем говорить, что слово w *читается* на пути p , если конкатенация меток вдоль p образует w .

Определение 0.3. *Контекстно-свободная грамматика — это четвёрка $\langle \Sigma, N, S, P \rangle$:*

- Σ — конечный алфавит
- N — конечное множество нетерминалов
- $S \in N$ — стартовый нетерминал
- P — конечное множество продукций (правил вывода), имеющих вид $N_i \rightarrow \alpha$, где $N_i \in N, \alpha \in (\Sigma \cup N)^*$

Язык, распознаваемый грамматикой \mathcal{G} — язык $L(\mathcal{G})$ слов, выводимых из стартового нетерминала.

Контекстно-свободная грамматика задаёт *контекстно-свободный язык*.

TODO: пример

TODO: в примере пояснить, что такое ε

Теперь определим саму задачу.

Определение 0.4. Дан ориентированный помеченный граф G и контекстно-свободная грамматика \mathcal{G} над тем же алфавитом.

Задача поиска путей с контекстно-свободными ограничениями (CFPQ)¹ заключается в нахождении всех пар вершин $u, v \in V(G)$, таких что существует путь $p : u \rightsquigarrow v$, на котором читается слово $w \in L(\mathcal{G})$.

TODO: пример

¹В реляционной семантике [24]

Цель и задачи

Целью работы является получить решения для частных случаев в задаче поиска путей с контекстно-свободными ограничениями, основываясь на едином подходе.

Для её достижения решаются следующие задачи:

- Выбрать единый подход для решения задачи CFPQ
- Построить на его основе решения для следующих частных случаев:
 - Язык Дика \mathcal{D}_k и двунаправленные графы
 - Язык Дика \mathcal{D}_1 (на одном типе скобок)
- Применить выбранного подхода для решение задачи достижимости для смешанный языка Дика $\mathcal{D}_1 \odot \mathcal{D}_1$ на двунаправленных графах

TODO: Всё тут переписать

Структура работы

В главе 1 приведена история развития области и анализ существующих решений для задачи CFPQ.

В главе 2 представлен выбранный подход к решению задачи CFPQ (идея пересечения языков).

В главе 3 описан алгоритм, основанный на неориентированном инкрементальном транзитивном замыкании, и доказана его корректность для двунаправленных графов и языка Дика.

В главе 4 описан алгоритм, основанный на неинкрементальном транзитивном замыкании, и его применение для языка Дика на одном типе скобок.

В главе 5 описан алгоритм для смешанного языка Дика.

В последней главе подведены итоги, а также описаны возможные направления будущего развития.

1. Обзор литературы

Задача поиска путей с контекстно-свободными ограничениями (она же CFPQ) была сформулирована Михалисом Яннакакисом ещё в 1990 году [54] в применении к запросам к декларативному языку Datalog [51, 9], textitно эта идея не была особо развита.

В последнее десятилетие интерес к задаче *вспыхнул с новой силой* в контексте запросов к RDF¹ [39] — графовой модели представления данных в сети, разработанной W3C². RDF хранит объекты (ресурсы) и утверждения об их связях (тройки «субъект — предикат — объект»). Чаще всего для работы с данными, представленными в RDF, используют язык запросов SPARQL [43]. Однако SPARQL позволяет осуществлять только запросы, представленные в виде регулярных выражений, тогда как некоторые интересные запросы (такие как *same-generation queries*³ [1]) могут быть выражены только в терминах контекстно-свободных языков.

Задача также нашла широкое применение в разных видах статического анализа [41] (в это области она более известная под именем задачи контекстно-свободной достижимости или CFL-Reachability), таких как

TODO: написать про виды стат анализа..

1.1. Решения задачи в общем случае

За более чем 30 лет было предложено множество алгоритмов для решения задачи CFPQ.

Большая часть решений реализует идеи различных алгоритмов разбора выражений (парсинга). Так, алгоритм Мельски и Репса [34] использует тот же подход, что и алгоритм Кока-Янгера-Касами [57] парсинга КС-языков: приведение грамматики к нормальной форме Хомского [15] и подсчёт динамического программирования — и имеет ту же асимптотику $\mathcal{O}(|V|^3|N|^3)$, где $|V|$ — число вершин в графе, $|N|$ — число нетерминалов входной грамматики. Позднее Чаудхури [12] улучшил этот алгоритм, уменьшив асимптотику в $\log |V|$ раз, используя метод четырёх русских [37].

Алгоритм Григорьева и Рогозиной, основан на обобщённом нисходящем синтаксическом анализе⁴ — GLL [45] парсинге, и работает за $\mathcal{O}(|V|^3 \max_{v \in V} \deg^+(v))$. Алгоритм Медейроса и др. [33] также основан на нисходящем синтаксическом анализе — LL парсинге, но имеет время работы $\mathcal{O}(|V|^3|P|)$, где $|P|$ — число продукций входной грамматики.

Алгоритм Сантоса и др. [44] основан на восходящем синтаксическом анализе⁵ —

¹Resource Description Framework — Среда описания ресурса

²World Wide Web Consortium — Консорциум Всемирной паутины

³Запросы поиска объектов, находящихся на одном уровне иерархии

⁴Top-down parsing

⁵Bottom-up parsing

Tomita-Style Generalized LR парсинге [46], однако его время работы не оценено (**TODO**: точно?), а на некоторых входах он вообще не завершается (однако на практике показывает хорошую производительность).

Но есть и подходы, не основанные на алгоритмах парсинга. Например, в своей работе Азимов и Григорьев [44] сводят задачу CFPQ к транзитивному замыканию матриц (по аналогии с решением Валианта [50] задачи распознавания КС-языков). Преимущество этого алгоритма в том, что он использует операции над матрицами, которые могут быть оптимизированы с использованием GPGPU¹.

Хеллингс в своей работе [23] рассматривает задачу в основанной на путях (path-based) семантике запроса и разрешает её, используя аннотированные грамматики.

Чаудхури [13], а также Орачев и др. [16] сводят задачу CFPQ к задаче достижимости в рекурсивном конечном автомате, которую решают, используя инкрементальное транзитивное замыкание. Наивная реализация работает за $\mathcal{O}(|V|^3|N|^3)$, но так же (как и алгоритм Репса [34, 12]) может быть оптимизирован [38] в $\log |NV|$ раз методом четырёх русских.

1.2. Нижние оценки

Как можно заметить, все существующие алгоритмы (для решения CFPQ в общем случае) имеют кубическое (или большее) время работы. Что не достаточно эффективно для работы с реальными данными, как экспериментально показали Кёйперс и др. [19], реализовав и замерив производительность трёх алгоритмов: Хеллингса [23], Сантоса и др. [44] и Азимова и др. [44].

Более того, скорее всего, решения с более быстрой ($\mathcal{O}(|V|^{3-\epsilon})$) асимптотикой не существует. Это так называемый “cubic bottleneck”² [22] данной задачи. Было доказано, что она является 2NPDA³-полной, и субкубическое решение для неё повлечёт наличие субкубических алгоритмов для всех задач класса. Учитывая, что такие решения не были найдены за более чем 50 лет, маловероятно, что данная задача решается быстрее куба.

Существуют и другие условные нижние оценки.

Так, Чаттерджи и др. в своей работе [10] построили условную нижнюю оценку, сведя в к задаче CFPQ (а именно, к \mathcal{D}_k -достижимости (опр. 1.2)) задачу ВММ⁴, на которую есть условная нижняя оценка. А именно, согласно ВММ-гипотезе [52], не существует субкубического *комбинаторного*⁵ алгоритма для перемножения двух булевых

¹General-purpose computing on graphics processing units — техника использования графического процессора для неграфических целей (математических вычислений)

²узкое место

³2-way nondeterministic pushdown automata [3] — 2-сторонний автомат с магазинной памятью

⁴Boolean Matrix Multiplication — перемножение двух булевых матриц

⁵Этот термин не вполне определен, но можно понимать его как “не алгебраический”. В частности, комбинаторные алгоритмы не должны использовать деление и вычитание, так те пользуются особен-

матриц. Замечание про комбинаторность алгоритма важно, так как алгебраическое субкубическое решения для ВММ существует, а именно, она сводится к обычному перемножению матриц, которое может быть совершено за $\mathcal{O}(|V|^\omega)^1$.

Позднее Чжан [61] улучшил эту оценку, построив сведение ВММ к \mathcal{D}_1 -достижимости, тем показав, что, скорее всего, не существует субкубического комбинаторного решения уже для неё.

И если с комбинаторными алгоритмами всё более менее понятно и оценки (нижняя и верхняя) сходятся, то с некомбинаторными всё не так радужно. Нижняя оценка в $\mathcal{O}(|V|^\omega)$ вытекает из сведения от ВММ, но обратного сведения не построено и непонятно, может ли эта оценка быть достигнута. Более того, существует условная нижняя оценка на нижнюю оценку: Чистиков и др. [14] показали, что при условии NSETH² [36] не существует нижней оценки, основанной на SETH³ [28] для \mathcal{D}_2 -достижимости, лучшей, чем $\mathcal{O}(|V|^\omega)$.

1.3. Решения задачи в частных случаях

Понятно, что для решения практических задач далеко не всегда нужна CFPQ в общем случае. Чаще всего для каждой конкретной задачи нужна конкретная КС грамматика, а иногда ещё и понятны ограничения на тип графа.

Пользуясь этой информацией (ограничениями на тип грамматики и графа) можно конструировать частные, более быстрые решения, или давать более точные оценки на время работы существующих.

Начнём с очевидных частных случаев. Так, для графа-цепочки задача CFPQ — это просто задача КС-распознавания, так как граф-цепочка — это просто одна строка. А для задачи КС-распознавания существует субкубическое решение [50], более того, она эквивалентна задаче перемножения булевых матриц (так что решается за $\mathcal{O}(|V|^\omega)$). В своей работе [54] Яннакакис также заметил, что это сведение можно обобщить на случай ациклических графов.

Решение с такой же асимптотикой $\mathcal{O}(|V|^\omega)$ получено для ещё одного частного случая — иерархических грамматик⁴ [55]. На самом деле такие грамматики задают регулярные языки, однако размер автомата может быть экспоненциален относительно размера грамматики. Но если считать размер грамматики константным (как делают довольно часто), то задача поиска путей с регулярными ограничениями решается за $\mathcal{O}(V^\omega)$, так как сводится [54] к построению транзитивного замыкания входного графа (опр. **TODO**).

ностями алгебраических структур (а именно, существованием обратного)

¹ $2 < \omega < 2.373$ [4]

²Nondeterministic Strong Exponential Time Hypothesis — Недетерминированная сильная гипотеза об экспоненциальном времени

³Strong exponential time hypothesis — Сильная гипотеза об экспоненциальном времени

⁴Hierarchical state machines

Ещё одним из языков, для которых строятся частичные решения, является язык Дика.

Определение 1.1. Языком Дика¹ на k типах скобок \mathcal{D}_k называют язык, заданный над алфавитом $\Sigma_k = \{(1,)_1, (2,)_2 \dots (k,)_k\}$ и состоящий из правильных скобочных последовательностей на k типах скобок.

Язык Дика — контекстно-свободный и задаётся следующей грамматикой:
 $\mathcal{D}_k : S ::= SS \mid (1S)_1 \mid (2S)_2 \mid \dots (kS)_k \mid \varepsilon$

Определение 1.2. Задачу CFPQ для языка Дика называют также задачей Диковой достижимости² [30] или \mathcal{D}_k -достижимости.

Задача Диковой достижимости широко применяется в статическом анализе, так как во многих видах анализа возникают парные объекты: вызовы и возвраты из функций [48], обращение по указателю и их разыменование [62], запись и чтение из поля [53], взятие и возврат блокировки [29].

Первыми улучшениями для задачи Диковой достижимости стали лучшие оценки на время работы уже существующих алгоритмов. Так, Кодумал и Айкен [30] показали, что алгоритм Репса [34], применённый для грамматики Дика (размер которой $\mathcal{O}(k)$ для языка \mathcal{D}_k), имеет асимптотику $\mathcal{O}(|V|^3 k)$, тогда как обычная оценка — $\mathcal{O}(|V|^3 k^3)$. А Рехоф и Фендрих [40] показали оценку (также для алгоритма Репса) в $\mathcal{O}(|V|^3)$ для языка Дика и графов потока исполнения, построенных для решения задачи основанного на типах анализа потока³.

Как уже было сказано выше, существует кубическая нижняя оценка на время работы комбинаторного алгоритма уже для задачи Диковой достижимости. Это однако не мешает существованию субкубических алгебраических алгоритмов. Так, для задачи достижимости для языка Дика на одном типе скобок \mathcal{D}_1 были построены более быстрые алгоритмы. Бредфорд сводит [8] задачу \mathcal{D}_1 -достижимости к $\mathcal{O}(\log^2 |V|)$ перемножениям AGMY-матриц⁴, каждое из которых может быть произведено за $\mathcal{O}(|V|^\omega \log |V|)$. Матиасен и др. [32] строят “более комбинаторное” решения, строя сведение \mathcal{D}_1 -достижимости к $\mathcal{O}(\log^2)$ непосредственно перемножений булевых матриц (для подсчёта транзитивного замыкания графа [2]).

В общем же случае (для языка Дика более, чем на одном типе скобок) субкубических решений не существует. Однако они появляются при добавлении ограничений на вид графа. А именно, при рассмотрении двунаправленных графов.

Определение 1.3. Помеченный граф $G = \langle V, E, \Sigma_k \rangle$ называют двунаправленным⁵ [58],

¹Dyck language

²Dyck-reachability

³Type-based flow analysis

⁴Alon, Galil, Margalit [5]; и Yuval [59]

⁵Bidirected graph

если в нём для каждого ребра $\langle u, v, (i) \rangle$ найдётся противоположное ребро $\langle v, u,)_i \rangle$ и наоборот.

Неформально, матрица смежности такого графа симметрична, и метки на симметричных рёбрах — это парные открывающая/закрывающая скобки.

TODO: картиночка

В своей работе Юань и др. [58] заметили, что при решении задачи анализа указателей¹ граф получается двунаправленным (так как отношения `GetField` и `PutField` взаимнообратные). Также, они конструируют алгоритм, с временем работы $\mathcal{O}(|V| \log |V| \log k)$, в случае, если полученный граф является деревом. На внешнем уровне алгоритма строится центроидная декомпозиция, а для каждого конкретного центроида — бор S -префиксов² его поддеревя. Чжан и др. [20], заметив, что в двунаправленных графах Дикова достижимые вершины формируют классы эквивалентности (3.1), улучшили этот алгоритм, получив линейное время работы, а также сконструировали алгоритм для произвольных графов с временем работы $\mathcal{O}(|E| \log |E|)$. Позднее, Чаттерджи и др. [10] улучшили этот результат до $\mathcal{O}(|E| + |V|\alpha(|V|))$ ³.

Если к задаче анализа указателей добавить контекстную чувствительность⁴, то есть кроме записи/чтения полей учитывать ещё и вызов/возврат из функций, то её можно сформулировать в терминах смешанной Диковой достижимости.

Определение 1.4. Для двух языков L_1 и L_2 , заданных над алфавитами Σ_1 и Σ_2 соответственно, определим *оператор смешения*⁵ [31] $\odot : L_1 \times L_2 \rightarrow (\Sigma_1 \cup \Sigma_2)^*$ следующим образом:

- $a \odot \varepsilon = \{a\}$, где $a \in L_1$
- $\varepsilon \odot b = \{b\}$, где $b \in L_2$
- $c_1 a \odot c_2 b = \{c_1 w \mid w \in (a \odot c_2 b)\} \cup \{c_2 w \mid w \in (c_1 a \odot b)\}$,
где $a \in L_1, b \in L_2, c_1 \in \Sigma_1, c_2 \in \Sigma_2$

Можно также переопределить оператор смешения для двух языков:

$$L_1 \odot L_2 = \bigcup_{a \in L_1, b \in L_2} a \odot b.$$

Определение 1.5. Пусть есть два языка Дика \mathcal{D}_i и \mathcal{D}_j , заданные над разными алфавитами. Тогда назовём язык $\mathcal{D}_i \odot \mathcal{D}_j$ *смешанным языком Дика*⁶.

Неформально, это множество таких скобочных последовательностей, что их проекции на алфавиты Σ_i и Σ_j принадлежат \mathcal{D}_i и \mathcal{D}_j соответственно.

¹Points-to Analysis

²Строк, сформированных наивным алгоритмом проверки правильности скобочной последовательности с использованием стека

³ $\alpha(n)$ — обратная функция Аккермана

⁴Context sensitivity

⁵Interleaving operator

⁶Interleaved Dyck language или INTERDYCK language

Например, пусть \mathcal{D}_b — языка квадратных ПСП и \mathcal{D}_p — языка круглых ПСП. Тогда смешанный язык $\mathcal{D}_b \odot \mathcal{D}_p$ содержит такие слова как “()” и “([()] [])”.

Однако, задача смешанной Диковой достижимости в общем случае неразрешима [42]. Более того, она неразрешима уже для языка $\mathcal{D}_2 \odot \mathcal{D}_2$ даже на двунаправленных графах. Поэтому Ли и др. [31] рассмотрели задачу $\mathcal{D}_1 \odot \mathcal{D}_1$ -достижимости (на двунаправленных графах) и построили для неё алгоритм с временем работы $\mathcal{O}(|V|^7)$. Решение этой задачи можно использовать как приближение для более общей $\mathcal{D}_k \odot \mathcal{D}_k$ -достижимости.

1.4. Выводы и результаты по главе

TODO



2. Алгоритм, основанный на пересечении языков

В качестве единого подхода к построению частичных решений для задачи CFPQ был выбран алгоритм [13, 16], основанный на пересечении языков. Причина выбора именно этого решения следующая: в основе остальных подходов лежат алгоритмы парсинга, все из которых имеют кубическое время работы и неизвестно, могут ли быть ускорены, тогда как данный подход основан на графовых алгоритмах (а именно, на решении задачи построения инкрементального транзитивного замыкания), засчёт чего более подвержен модификациям.

2.1. Сведение к задаче достижимости для РКА

Главной идеей алгоритма является следующее замечание: любой помеченный граф G можно рассматривать как *недетерминированный конечный автомат*, в котором не обозначены начальное и конечные состояния.

Определение 2.1. *Недетерминированный конечный автомат (или НКА)*¹ — это пятёрка $\langle Q, \Sigma, \delta, q_0, F \rangle$:

- Q — конечное множество состояний
- Σ — конечный алфавит
- $\delta: Q \times \Sigma \rightarrow 2^Q$ — функция перехода
- $q_0 \in Q$ — начальное (стартовое) состояние
- $T \subseteq Q$ — множество конечных (терминальных) состояний

Язык, распознаваемый НКА \mathcal{A} — язык $L(\mathcal{A})$ слов, на которых автомат, следуя функции перехода, может дойти из стартового состояния в терминальное хотя бы одним способом.

Недетерминированные конечные автоматы задают *регулярные языки*.

TODO: пример (в фигуру с автоматом $0 \rightsquigarrow 2$)

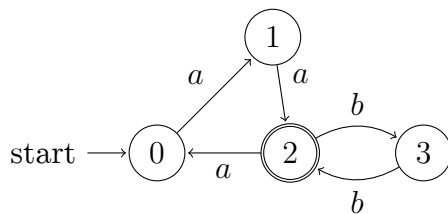


Рис. 3: НКА, задающий слова, читаемые на путях $0 \rightsquigarrow 2$.
Например, aa , $aabb$ или $aabbbaaa$

¹Nondeterministic finite automaton (или NFA)

Утверждение 2.1. Если зафиксировать конкретные вершины u и v помеченного графа G как стартовое и конечное состояния, то полученный автомат будет задавать язык слов, читаемых на путях из u в v .

Язык, задаваемый подобным автоматом, будем обозначать как $L(G, u, v)$.

Получаем, что для каждой пары вершин u и v , наличие пути $p: u \rightsquigarrow v$, такого, что на нём читается слово $w \in L(\mathcal{G})$ равносильно непустоте пересечения языков $L(G, u, v)$ и $L(\mathcal{G})$ (действительно, в пересечении будут лежать ровно слова, выводимые грамматикой и при этом читаемые на каком-либо пути $u \rightsquigarrow v$).

Для построения пересечения языков нам потребуется такая конструкция, как прямое произведение автоматов.

Определение 2.2. Для двух НКА $\mathcal{A}_1 = \langle Q_1, \Sigma_1, \delta_1, q_{01}, T_1 \rangle$ и $\mathcal{A}_2 = \langle Q_2, \Sigma_2, \delta_2, q_{02}, T_2 \rangle$ прямым произведением $\mathcal{A} = \mathcal{A}_1 \otimes \mathcal{A}_2$ назовём НКА $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, T \rangle$, такой что:

- $Q = Q_1 \times Q_2$ (т.е. состояние в \mathcal{A} — пара состояний в \mathcal{A}_1 и \mathcal{A}_2)
- $\Sigma = \Sigma_1 \cap \Sigma_2$
- для переходов $s_1 \xrightarrow{a} t_1 \in \delta_1$ и $s_2 \xrightarrow{a} t_2 \in \delta_2$ существует переход $\langle s_1, s_2 \rangle \xrightarrow{a} \langle t_1, t_2 \rangle$
- $q_0 = \langle q_{01}, q_{02} \rangle$
- $T = T_1 \times T_2$

Утверждение 2.2. [26] Автомат \mathcal{A} , построенный как прямое произведение автоматов \mathcal{A}_1 и \mathcal{A}_2 , распознаёт язык, равный пересечению языков \mathcal{A}_1 и \mathcal{A}_2 .

Заметим, однако, что лишь один из языков, чьё пересечение нам нужно, является регулярным (и, следовательно, задаётся автоматом) — язык входного графа. Второй же язык является контекстно-свободным и недетерминированный конечным автоматом не задаётся. Однако он задаётся с помощью более сложного автомата — рекурсивного.

Определение 2.3 (Рекурсивный конечный автомат (РКА)). Рекурсивный конечный автомат (или РКА)¹ [6] — это набор компонент $\langle M_1, M_2, \dots, M_k \rangle$, с выделенной стартовой компонентой, где каждая компонента M_i — это пятёрка $\langle Q_i, \Sigma_i, En_i, Ex_i, \delta_i \rangle$:

- Q_i — конечное множество состояний
- Σ_i — конечный алфавит
- $En_i \subset Q_i$ — множество начальных состояний
- $Ex_i \subset Q_i$ — множество конечных состояний
- $\delta_i: Q_i \times (\Sigma_i \cup \bigcup_{j=1}^k En_j \times Ex_j) \rightarrow 2^{Q_i}$ — функция перехода. У δ_i есть два типа переходов: *внутренние*, которые работают как обычные переходы в НКА и *рекурсивные*, которые делают вызов другой компоненты (при этом обозначая начальную и конечную вершину в ней).

¹Recursive state machine (или RSM)

Неформально, это набор компонент, каждая из которых представляет собой НКА, на рёбрах которого могут быть “рекурсивные вызовы” других компонент.

TODO: пример

TODO: показать, как строится РКА по грамматике

Утверждение 2.3. [6] *Утверждение 2.2 остаётся верным, если один из языков задан РКА.*

При прямом произведении НКА и РКА получается также РКА. А именно, при произведении РКА $\mathcal{R} = \langle M_1, \dots, M_k \rangle$, $M_i = \langle Q_i, \Sigma_i, En_i, Ex_i, \delta_i \rangle$ и НКА $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$, получается РКА $\mathcal{P} = \mathcal{R} \otimes \mathcal{A}$, состоящий из k компонент \mathcal{M}_i , таких что

- множество состояний \mathcal{M}_i равно декартову произведению $Q_i \times Q$
- множество начальных состояний \mathcal{M}_i равно $En_i \times Q$
- множество конечных состояний \mathcal{M}_i равно $Ex_i \times Q$
- для каждого внутреннего перехода $q_s \xrightarrow{a} q_t \in \delta_i$, в \mathcal{M}_i есть внутренние переходы $\langle q_s, u \rangle \xrightarrow{a} \langle q_t, v \rangle$ для всех $u, v \in Q$, что $u \xrightarrow{a} v \in \delta$
- для каждого внешнего перехода $\langle q_s \xrightarrow{en, ex} q_t \rangle \in \delta_i$, в \mathcal{M}_i есть внешние переходы $\langle q_s, q \rangle \xrightarrow{\langle en, q \rangle, \langle ex, q \rangle} \langle q_t, q \rangle$ для всех $q \in Q$

Для проверки непустоты получившегося языка, нужно проверить, существует ли путь из стартового состояния $\langle q_s, u \rangle$ в конечное состояние $\langle q_t, v \rangle$ (q_s и q_t принадлежат начальной компоненте). То есть решить задачу достижимости для РКА.

Как будет обсуждено позже, при решении задачи достижимости для РКА, строится матрица достижимости (т.е. для каждой пары состояний находится, есть ли путь из одного в другое). Так что за один запуск такого алгоритма можно проверить КС-достижимость для всех пар вершин исходного графа.

Подводя итог, все алгоритмы для CFPRQ, основанные на идее пересечения языков, будут иметь следующую схему:

1. Построить прямое произведение входной грамматики \mathcal{R} и входного графа G : $\mathcal{P} = \mathcal{R} \otimes G$.
2. Решить задачу достижимости для полученного РКА \mathcal{P}
3. Из вершины u в вершину v входного графа существует путь, выводимой входной грамматикой $\mathcal{G} \Leftrightarrow$ в \mathcal{P} есть путь $(q_0, u) \rightsquigarrow (q_f, v)$ из стартового в конечное состояние начальной компоненты РКА \mathcal{P}

Второй пункт — решение задачи достижимости для РКА — будет подробно рассмотрен далее.

2.2. Решение задачи достижимости для РКА

Решение задачи достижимости для НКА эквивалентно [54] решению задачи достижимости для обычного ориентированного графа и заключается в построении транзитивного замыкания.

Определение 2.4. **TODO**

В случае РКА же всё сложнее — про часть рёбер (все рекурсивные рёбра) непонятно, присутствуют ли они в графе. А именно, рекурсивное ребро $u \xrightarrow{en,ex} v$ присутствует (может быть использовано для построения путей) только если состояние ex достижимо из состояния en .

Для обработки таких ситуаций задачу решают итеративно: при обнаружении достижимости пары вершин проводят новые рёбра (рекурсивные рёбра с соответствующей меткой) и обновляют отношение достижимости (после этого могут появиться новые рёбра, которые снова проводят, и так далее). Такая задача уже сводится к решению задачи поддержания **инкрементального** транзитивного замыкания.

Определение 2.5. **TODO**

В листинге 1 приведён псевдокод алгоритма решения задачи достижимости для РКА, основанный на поддержании инкрементального транзитивного замыкания.

Замечание. Отдельно, до начала основного алгоритма, нужно обработать ε -переходы. Эта часть алгоритма опущена (здесь и далее) для простоты изложения.

Listing 1 Алгоритм достижимости для РКА

```

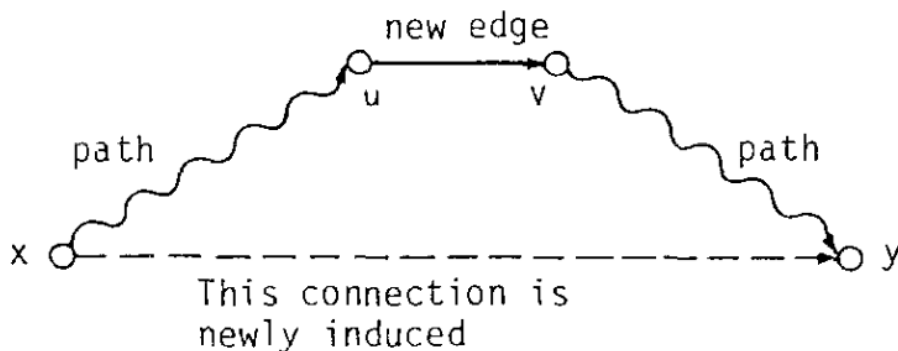
1: function RSMREACHABILITY2( $\mathcal{R}$ )
2:    $A \leftarrow$  Empty adjacency matrix
3:    $Q \leftarrow$  Empty Queue
4:   for  $i \in 1..k$  do
5:     for  $u \xrightarrow{c} v \in \delta_i$  do
6:        $Q.Push(\langle u, v, i \rangle)$ 
7:   while  $Q$  is not Empty do
8:      $\langle u, v, i \rangle \leftarrow Q.Pop()$ 
9:     if  $u \in En_i \wedge v \in En_i$  then ▷ Нашли новый путь
10:       $A \leftarrow A \cup getEdges(i, u, v)$ 
11:       $Q.PushAll(getEdges(i, u, v))$  ▷ Добавляем новые рёбра
12:     for  $x \in Q_i$  do
13:       if  $A_{x,u} \wedge \overline{A_{x,v}}$  then
14:         for  $y \in Q_i$  do
15:           if  $A_{v,y} \wedge \overline{A_{x,y}}$  then
16:              $A \leftarrow A \cup \langle x, y \rangle$ 
17:              $Q.Push(\langle x, y, i \rangle)$  ▷ Обновлем транзитивное замыкание
18:   return  $A$ 

```

Работа происходит над матрицей смежности \mathcal{R} — изначально туда записываются все внутренние (нерекурсивные) рёбра.

В алгоритме используется стандартная реализация инкрементального транзитивного замыкания [27]. Для этого в ходе работы алгоритма поддерживается рабочая очередь Q рёбер транзитивного замыкания, которые были найдены, но ещё не обработаны.

При обработке очередного ребра, ищутся новые пути, которые проходят через него. А именно, пусть было добавлено ребро $u \rightarrow v$. Тогда далее перебирается вершина x , такая что из неё была достижима вершина u ($x \rightsquigarrow u$), но не была достижима вершина v ($x \not\rightsquigarrow v$). Из такой вершины x становятся достижимы все вершины y , которые были достижимы из v ($v \rightsquigarrow y$).



TODO: норм картинка

Если новое ребро ТЗ (= путь в графе) соединяет начальную и конечную вершину, в очередь добавляются также все соответствующие ему рекурсивные рёбра: для нового пути $(en \in En_i) \rightsquigarrow (ex \in Ex_i)$ проводятся все рёбра, соответствующие рекурсивным вызовам i -ой компоненты с начальной вершиной en и конечной вершиной ex .

Время работы

Внешний цикл итерируется по всем рёбрам, так что работает за $\mathcal{O}(|E^*|)$ (где E^* — рёбра итогового транзитивного замыкания). Добавление новых рёбер рассмотрит каждое ребро не более одного раза, так что тоже работает за $\mathcal{O}(|E^*|)$. Нужно только оценить работу по поддержанию транзитивного замыкания.

Цикл на строке 12 перебирает все вершины, так что отработает суммарно за $\mathcal{O}(|E^*||V|)$. Внутренний цикл (строка 14) тоже перебирает все вершины, но после его выполнения будет добавлено хотя бы одно новое ребро $(x \rightarrow v)$, так что суммарное время работы этих циклов также можно оценить как $\mathcal{O}(|E^*||V|)$.

Итого, суммарное время работы составляет $\mathcal{O}(|E^*||V|)$, что в плотных графах будет равно $\Theta(|V|^3)$.

Замечание. Время работы инкрементального транзитивного замыкания можно ускорить в $\log |V|$ раз [38], воспользовавшись методом четырёх русских [37] (так как работа происходит над булевыми векторами).

2.3. Получение решений для частных случаев

Основой для получения частных решений будут модификации данного алгоритма 1.

Как уже было сказано, узким местом алгоритма является поддержание инкрементального транзитивного замыкания. На его время работы (так же, как и на время работы задачи CFPQ) существует кубическая условная нижняя оценка: к ней сводится [49] задача OMV¹.

Следовательно, чтобы получить более быстрые алгоритмы, нужно рассматривать такие частные случаи, для которых задачу инкрементального транзитивного замыкания можно решать за субкубическое время.

Опишем такие случаи:

- *Графы с ограниченной степенью*

В [56] представлен алгоритм для инкрементального транзитивного замыкания на графах с ограниченной исходящей степенью. Асимптотика алгоритма: $\mathcal{O}(d|E^*|)$, где d — ограничение сверху на исходящую степень графа.

- *Планарные графы*

TODO: Разобраться, что там написано у Александры

- *Автоматы, с ограниченным стеком² [13]*

Для автоматов, РКА которых не уходит в рекурсию, был построен [13] алгоритм, с временем работы $\mathcal{O}(|V|^3/\log^2 |V|)$.

- *Неориентированные графы*

Подробнее рассмотрены в главе 3

- **TODO:** ???

Подробно рассмотрены в главе 4

¹Online Matrix-Vector Multiplication — онлайн умножения матрицы и векторов

²Bounded-stack State Machines

2.4. Выводы и результаты по главе



TODO

3. Результаты (1)

В данной главе будет рассмотрена модификация алгоритма 1, основанная на ослаблении условия на граф, а именно, на “предположении”, что он является неориентированным. Также будет доказана корректность данной модификации для двунаправленных графов и языка Дика, а также для неориентированных графов и некоторых видов грамматик.

Для неориентированных графов отношение достижимости равно отношению “принадлежать одной компоненте связности”. Поддерживать добавление рёбер и проверку связности в неориентированном графе можно с помощью структуры данных Система Непересекающихся Множеств.

Определение 3.1. Система Непересекающихся Множеств (или СНМ)¹ [21] — структура данных для работы с непересекающимися множествами. Её интерфейс включает следующие операции:

- $MakeSet(x)$ — создаёт новое множество, состоящее из одного элемента x
- $Find(x)$ — возвращает элемент-представитель множества, содержащего x . Для всех элементов одного множества элемент-представитель одинаков
- $Union(x, y)$ — объединяет множества, содержащие элементы x и y

3.1. Алгоритм, основанный на неориентированном транзитивном замыкании

В листинге 2 приведён псевдокод алгоритма.

Listing 2 Алгоритм достижимости для РКА, основанный на неориентированном ИТЗ

```

1: function UNDIRECTEDRSMREACHABILITY( $\mathcal{R}$ )
2:    $A \leftarrow$  Empty adjacency matrix
3:    $Q \leftarrow$  Empty Queue
4:    $D \leftarrow$  DSU( $|\bigcup_{i=1}^k Q_i|$ )
5:   for  $i \in 1..k$  do
6:     for  $u \xrightarrow{c} v \in \delta_i$  do
7:        $Q.Push(\langle u, v, i \rangle)$ 
8:   while  $Q$  is not Empty do
9:      $\langle u, v, i \rangle \leftarrow Q.Pop()$ 
10:    if  $u \in E_{n_i} \wedge v \in E_{n_i}$  then ▷ Нашли новый путь
11:       $A \leftarrow A \cup getEdges(i, u, v)$ 
12:       $Q.PushAll(getEdges(i, u, v))$ 
13:       $D.Union(u, v)$  ▷ Добавляем новые рёбра
14:   return  $A$ 

```

¹Disjoint-set/union-find data data structure

Для реализации используются две вспомогательные структуры данных: очередь Q , хранящая рёбра, которые были добавлены в граф, но ещё не обработаны (как и в оригинальном алгоритме), и СНМ D , поддерживающая компоненты связности и поиск новых путей (стартовое состояние \rightarrow конечное состояние).

Опишем подробно структуру используемой СНМ (в листинге 3 приведён псевдокод).

Listing 3 Система Непересекающихся Множеств

```

1: Structure DisjointSets
2:   function DISJOINTSETS( $V$ )
3:     for  $v \in V$  do
4:        $P[v] \leftarrow v$                                 ▷ Предок
5:        $R[v] \leftarrow 0$                                 ▷ Ранг
6:     for  $v \in En(V)$  do
7:        $En[v] \leftarrow \{v\}$                             ▷ Список стартовых вершин поддева
8:     for  $v \in Ex(V)$  do
9:        $Ex[v] \leftarrow \{v\}$                             ▷ Список конечных вершин поддева
10:    function FIND( $v$ )
11:      if  $P[v] = v$  then return  $v$ 
12:      return  $P[v] = Find(P[v])$                         ▷ Эвристика сжатие путей
13:    function UNION( $u, v$ )
14:       $u \leftarrow Find(u)$ 
15:       $v \leftarrow Find(v)$ 
16:      if  $u = v$  then return
17:      if  $R[u] > R[v]$  then
18:         $Swap(u, v)$                                     ▷ Ранговая эвристика
19:       $Q.PushAll(\{\langle en_u, ex_v \rangle \mid en_u \in En[u], ex_v \in Ex[v]\})$ 
20:       $Q.PushAll(\{\langle en_v, ex_u \rangle \mid en_v \in En[v], ex_u \in Ex[u]\})$   ▷ Добавление новых
    рёбер
21:       $En[v] \leftarrow En[v] \cup En[u]$ 
22:       $Ex[v] \leftarrow Ex[v] \cup Ex[u]$ 
23:       $R[v] \leftarrow \max(R[v], R[u] + 1)$ 
24:       $P[u] = v$                                           ▷ Объединение компонент

```

За основу взята стандартная реализация [25] на подвешенных деревьях, использующая обе эвристики: сжатие путей и ранговую.

Дополнительно в корнях хранятся списки всех начальных и конечных состояний компоненты. При добавлении ребра в операции *Union* перебираются все пары начальная/конечная вершина из двух компонент и соответствующие им рёбра добавляются в рабочую очередь Q .

TODO: (подумать) можно ли добавлять сразу много рёбер и сжимать их дфсом (как Боровка)?

Время работы

Теорема 3.1. На РКА из $|V|$ состояний и $|E^*|$ рёбрах в транзитивном замыкании, алгоритм 2 отработает за время $\mathcal{O}(|V| + |E^*|\alpha(|V|))$

Доказательство. Как и в алгоритме 1 проход по внешнему циклу и добавление новых рекурсивных рёбер отработает за $\mathcal{O}(|E^*|)$, так как каждое ребро транзитивного замыкания рассмотрится не более одного раза.

Осталось оценить время работы $|E^*|$ вызовов функции *Union*. Каждый из них совершает 2 вызова функции *Find*, каждый из которых отработает за амортизированное $\mathcal{O}(\alpha(|V|))$. Также, если вершины находились в разных компонентах, происходит перебор всех пар начальное-конечное состояние. Однако, этот перебор суммарно переберёт каждое ребро транзитивного замыкания не более одного раза, так что суммарно отработает за $\mathcal{O}(|E^*|)$.

Итого, время работы алгоритма составляет $\mathcal{O}(|V| + |E^*|\alpha(|V|))$ ($\mathcal{O}(|V|)$ берётся из создания СНМ на $|V|$ множеств). \square

3.2. Корректность для неориентированных графов и некоторых классов грамматик

TODO: может, в мусорку этот subsection?

3.3. Корректность для двунаправленных графов и языка Дика

Теорема 3.2. Алгоритм 2 работает корректно на двунаправленных графах и языке Дика.

Для доказательства потребуется следующее вспомогательное утверждение:

Лемма 3.1. Для вершин двунаправленных графов отношения Диковой достижимости является отношением эквивалентности.

Доказательство. Действительно, если инвертировать (заменить все $($ на $)$ и наоборот) и развернуть правильную скобочную последовательность, то получится так же правильная скобочная последовательность.

Пример: ПСП: $'([] ())'$ \rightarrow развёрнутая: $')) [(' \rightarrow$ инвертированная: $' (() [])'$.

\square

Замечание. В доказательстве для простоты будет рассматриваться язык Дика \mathcal{D}_2 (на двух типах скобок). Все рассуждения обобщаются на случай k типов скобок.

Замечание. Для данного алгоритма будем использовать нестандартный РКА для языка Дика (изображён на рис. 4)

Рассмотрим путь p длины ≥ 4 , он уже не будет простым. Найдём первую *точку перегиба* пути — вершину, из которой путь идёт в состояние на слое с меньшим номером.

Внимательно посмотрев на РКА языка Дика, можно заметить, что все состояния имеют входящую степень 1 (для этого и нужен был нестандартный вид), так что рёбра, соседствующие в пути p с точкой перегиба имеют одинаковую метку и идут на один и тот же уровень. Обозначим эти рёбра как $\langle q_i, u \rangle \rightarrow \langle q_j, v \rangle \rightarrow \langle q_i, w \rangle$ ($\langle q_j, v \rangle$ — точка перегиба).

Рассмотрим все 3 варианта возможной метки на рёбрах вокруг точки перегиба.

- открывающая скобка $(_k$

$$p: (q_1, u) \rightarrow (q_{4/7}, v) \rightarrow (q_1, w) \rightarrow \dots \rightarrow (q_f, z).$$

Рёбра с меткой $(_k$ соответствуют рёбрам входного графа, а именно рёбрам $u \xrightarrow{(_k)} v$ и $w \xrightarrow{(_k)} v$. Заметим, что тогда, по двунаправленности графа, существует и ребро $v \xrightarrow{)_k} w$, дающее вместе с ребром $u \xrightarrow{(_k)} v$ путь $(_k)_k$ из u в w , порождающий S -ребро $\langle q_1, u \rangle \rightarrow \langle q_2, w \rangle$. Также, по предположению индукции существует простой путь $\langle q_1, w \rangle \rightarrow \langle q_f, z \rangle$, а значит есть и S -ребро $\langle q_2, w \rangle \rightarrow \langle q_3, z \rangle$. Вместе эти два ребра формируют просто путь $\langle q_1, u \rangle \rightarrow \langle q_2, w \rangle \rightarrow \langle q_3, z \rangle$, что и хотелось.

- S -метки

$$p: (q_1, a) \rightarrow \dots \rightarrow (q_i, u) \rightarrow (q_j, v) \rightarrow (q_i, w) \rightarrow \dots \rightarrow (q_f, z).$$

По лемме 3.1, из наличия дикого пути $w \rightsquigarrow v$ следует наличие дикого пути $v \rightsquigarrow w$. Вместе два диковых пути $u \rightsquigarrow v$ и $v \rightsquigarrow w$ образуют один путь $u \rightsquigarrow w$.

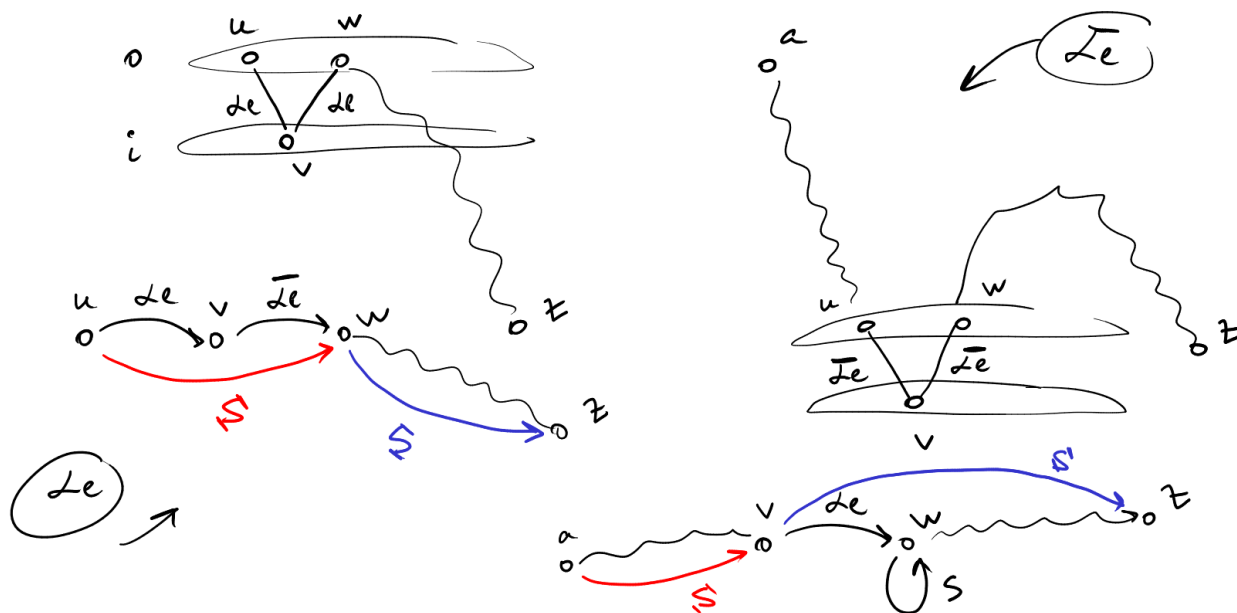
Сожмём этот путь в одну вершину uw . Получим более короткий путь, для него по предположению индукции существует аналогичный простой путь. Если этот простой путь не проходит через вершину uw , то он и является ответом. Иначе. Если вершина uw находится на 2, 4 или 7 слое, то следующее за ней ребро в простом пути имеет метку S , а значит, вместе с диковым путём $u \rightsquigarrow w$ собирается в один диков путь (по правилу $S \rightarrow SS$, как в случае $(_k)$). Если же uw находится на 1 слое, то весь путь от неё до z — один диков путь, который так же складывается с путём $u \rightsquigarrow w$.

- закрывающая скобка $)_k$

$$p: (q_1, a) \rightarrow \dots \rightarrow (q_i, u) \rightarrow (q_f, v) \rightarrow (q_i, w) \rightarrow \dots \rightarrow (q'_f, z).$$

По предположению индукции, существует простой путь $a \rightsquigarrow v$, так что есть S -ребро $\langle q_1, a \rangle \rightarrow \langle q_2, v \rangle$. Поймём, почему будет и ребро $\langle q_2, v \rangle \rightarrow \langle q_3, z \rangle$, которое вместе с предыдущим даёт искомый простой путь.

По двунаправленности графа, из наличия ребра $w \xrightarrow{k} v$ следует наличие ребра $v \xrightarrow{(k)} w$. Вместе с изначальным ε -ребром $w \xrightarrow{S} w$ и остатком пути $\langle q_i, w \rangle \rightsquigarrow \langle q'_f, z \rangle$ получаем более короткий путь, для которого, по предположению индукции существует аналогичный простой путь. Он и порождает второе ребро $\langle q_2, w \rangle \rightarrow \langle q_3, z \rangle$.



TODO: норм картинка

□

3.4. Выводы и результаты по главе



TODO

4. Результаты (2)

Вспомним алгоритм 1. Его слабым местом, дающим кубическую нижнюю оценку, было поддержание инкрементального транзитивного замыкания. Проблемы была в том, что рёбра могли добавляться по одному, и тогда никаким более простым/быстрым методом было не справиться. Однако, если новые пути появляются не по-одному, а сразу большими группами, задачу можно решать эффективнее.

4.1. Алгоритм, основанный на неинкрементальном транзитивном замыкании

Ключом к более быстрому решению является построение транзитивного замыкания с нуля на каждой итерации алгоритма. И, так как обычное транзитивное замыкание можно найти за субкубическое время, то если итераций алгоритма немного, такое решение получается асимптотически быстрее основного (алгоритма 1).

В листинге 4 приведён псевдокод решения.

Listing 4 Алгоритм достижимости для PKA

```
1: function RSMREACHABILITY( $\mathcal{R}$ )
2:    $A \leftarrow$  Adjacency matrix for  $\mathcal{R}$ 
3:   while  $A$  is changing do
4:      $A' \leftarrow \text{transitiveClosure}(A)$  ▷ Построение транзитивного замыкания
5:     for  $i \in 1..k$  do
6:       for  $u \in En_i$  do
7:         for  $v \in Ex_i$  do
8:           if  $A'_{u,v} \wedge \overline{A_{u,v}}$  then
9:              $A' \leftarrow A' \cup \text{getEdges}(i, u, v)$  ▷ Добавление новых рёбер
10:     $A \rightarrow A'$ 
11:  return  $A$ 
```

Изначально, так же, как и в алгоритме 1, в матрицу смежности A записываются все внутренние рёбра PKA \mathcal{R} .

Далее, внешний цикл повторяется, пока матрица смежности A меняется (т.е. пока добавляются новые рёбра). На каждой итерации считается A' — транзитивное замыкание A . После этого находятся все новые пути вида $\langle \text{стартовое состояние} \rangle \rightsquigarrow \langle \text{конечное состояние} \rangle$ — те рёбра между стартовой и конечной вершинами компоненты, которых не было в A , но которые есть в A' — и добавляются соответствующие этим путям рёбра.

Время работы

Время работы алгоритма — $k \cdot T(|V|)$, где k — число итераций внешнего цикла, $T(|V|)$ — время работы одной итерации.

Оценим $T(|V|)$. Внутренняя часть цикла состоит из двух частей: нахождения транзитивного замыкания (строка 4) и прохода по матрице для выявления новых рёбер (строки 5-9).

Задача поиска транзитивного замыкания эквивалентна задаче перемножения булевых матриц [2] и может быть решена сведением к быстрому перемножению (обычных) матриц за $\mathcal{O}(|V|^\omega)$.

Проход по матрице (строки 5-7) работает за $\mathcal{O}(|V|^2)$, что доминируется временем построения транзитивного замыкания. Добавление новых рёбер (строки 8-9) отрабатывает суммарно за $\mathcal{O}(|V|^2)$ (т.к. каждое ребро будет добавлено не более одного раза).

Итого, время работы алгоритма составляет $\mathcal{O}(k \cdot |V|^\omega)$.

4.2. Алгоритм для языка Дика на одном типе скобок

TODO: для semi-dyck тоже работает, нужно это упомянуть

Определение 4.1. Строки, принадлежащие языку Дика \mathcal{D}_1 часто изображают в виде *путей Дика* — путей из точки $(0, 0)$ в точку $(0, 2n)$, не опускающихся ниже оси абсцисс. Открывающей скобке соответствует вектор $(1, 1)$, закрывающей — $(1, -1)$.

TODO: Картинка

Части путей, образованные строками вида $(^k)^k$ будем называть *горами*.

Для построения алгоритма воспользуемся следующим результатом:

Лемма 4.1 (Дюлеаж и Лоран [17]). Для языка L определим $p_L(n)$ ¹ — максимальная длина кратчайшего слова в $L \cap K$ по всем регулярным языкам K , задаваемым НКА с $\leq n$ состояниями.

Тогда для языка Дика \mathcal{D}_1 на одном типе скобок $p_{\mathcal{D}_1}(n) = \mathcal{O}(n^2)^2$

Следствие 4.0.1. Для любой пары вершин $u, v \in V(G)$, если есть Диков путь $u \rightsquigarrow v$, то существует и Диков путь $u \rightsquigarrow v$, длина которого $\mathcal{O}(|V|^2)$.

Доказательство. Слова, читаемые на путях $u \rightsquigarrow v$ задаются НКА на n вершинах — графом G , в котором u и v выбраны за начальное и конечное состояния соответственно. □

Замечание. Пользуясь этим фактом, можно построить наивный алгоритм — достаточно лишь заметить, что раз длина искомого пути всегда ограничена, то можно задать такие пути с помощью автомата: язык \mathcal{D}_1 задаётся автоматом с одним счётчиком, значение счётчика не превышает $\mathcal{O}(|V|^2)$, так что его можно закодировать в состояние. Однако и размер такого автомата будет $\mathcal{O}(|V|^2)$, что для данной задач слишком много.

¹Формально, $p_L(n) = \max\{\min\{|w| : w \in L \cap K\} : K \in \text{Rat}_n(X), L \cap K \neq \emptyset\}$, где $\text{Rat}_n(X)$ — регулярные языки над алфавитом X , распознаваемые НКА с $\leq n$ состояниями.

²Точная оценка — $2n^2 + 4n$

Замечание. Вообще, для любого регулярного языка задача CFPQ решается построением обычного транзитивного замыкания от графа-произведения, то есть за $\mathcal{O}((|V||\mathcal{A}|)^\omega)$, где $|\mathcal{A}|$ — число состояний автомата.

Наивный алгоритм имеет такое большое время работы из-за большого размера автомата. Воспользовавшись же алгоритмом 4 можно построить более быстрое решение. Тем более известно, что часто размеры грамматик и автоматов для одних и тех же языков отличаются в экспоненциальное число раз.

Заметим, что для этого придётся видоизменить грамматику, для её стандартного вида $(\mathcal{D}_1 : S ::= SS|(S)|\varepsilon)$ алгоритм 4 может совершить до $\mathcal{O}(|V|^2)$ итераций: например, на цепочке $(^k)^k$ за одну итерацию будет проводиться лишь одно новое ребро (используя продукцию $S \rightarrow (S)$), так что потребуются все k итераций.

Другое представление языка \mathcal{D}_1 основано на двух ~~кита~~ замечаниях: длина максимального слова всего $\mathcal{O}(|V|^2)$, хочется “сжимать” длинные горы (4.1) быстрее, чем за их длину.

4.2.1. Грамматика для языка \mathcal{D}_1

Пусть $K = \lceil \log(2n^2 + 4n) \rceil$ (т.е. это логарифм длины максимального слова), построенная грамматика будет иметь размер $\mathcal{O}(K)$.

Грамматика будет содержать $2K$ нетерминалов, отвечающих за “сжатие” вертикальных путей: U_i “сжимает” пути вида $(^i, D_i$ сжимает пути вида $)^{2^i}$:

$$U_0 ::= (\tag{1}$$

$$U_1 ::= U_0 S U_0 \tag{2}$$

$$U_2 ::= U_1 S U_1 \tag{3}$$

$$\dots \tag{4}$$

$$U_K ::= U_{K-1} S U_{K-1} \tag{5}$$

В продукции для U_i есть S между U_{i-1} и U_{i-1} — она нужна, U_i ищет не строго возрастающие пути, а разрешает им некоторое время “идти прямо” (получаются уже такие пути Моцкина [18], а не Дика, в которых рёбра с метками S соответствуют горизонтальным рёбрам).

Продукции для D_i устроены аналогично.

Теперь можем строить нетерминал, отвечающую за, собственно, пути Дика. Продукция $S \rightarrow (S)$ в обычной грамматике как бы “сжимала” вершину горы. Теперь, пользуясь U_i и D_i она может “сжимать” горы побольше.

$$\mathcal{D}_1 : S ::= U_0 S D_0 \mid U_1 S D_1 \mid \dots \mid U_K S D_K \mid S S \varepsilon \tag{6}$$

Для того, чтобы число итераций алгоритма 4 было небольшим, РКА по данной грамматике нужно строить не совсем стандартным образом.

Так, продукция $S \rightarrow SS$ отвечает за транзитивное замыкание рёбер с S -метками. В случае РКА этого можно добиться, проведя S -петлю на терминальной вершине РКА (обозначим это как продукцию $S \rightarrow S^*$).

Также, при оценке асимптотики алгоритма будет важно, что все компоненты РКА имеют константный размер, так что компонента S будет разбита на K штук: $S_i ::= U_i S D_i$ для всех $0 \leq i \leq K$, а также $S ::= (S_0 \mid S_1 \mid \dots \mid S_k)^*$ (одно стартовое/терминальное состояние, на котором весит K петель), теперь эта компонента целиком отвечает только за транзитивное замыкание S -меток.

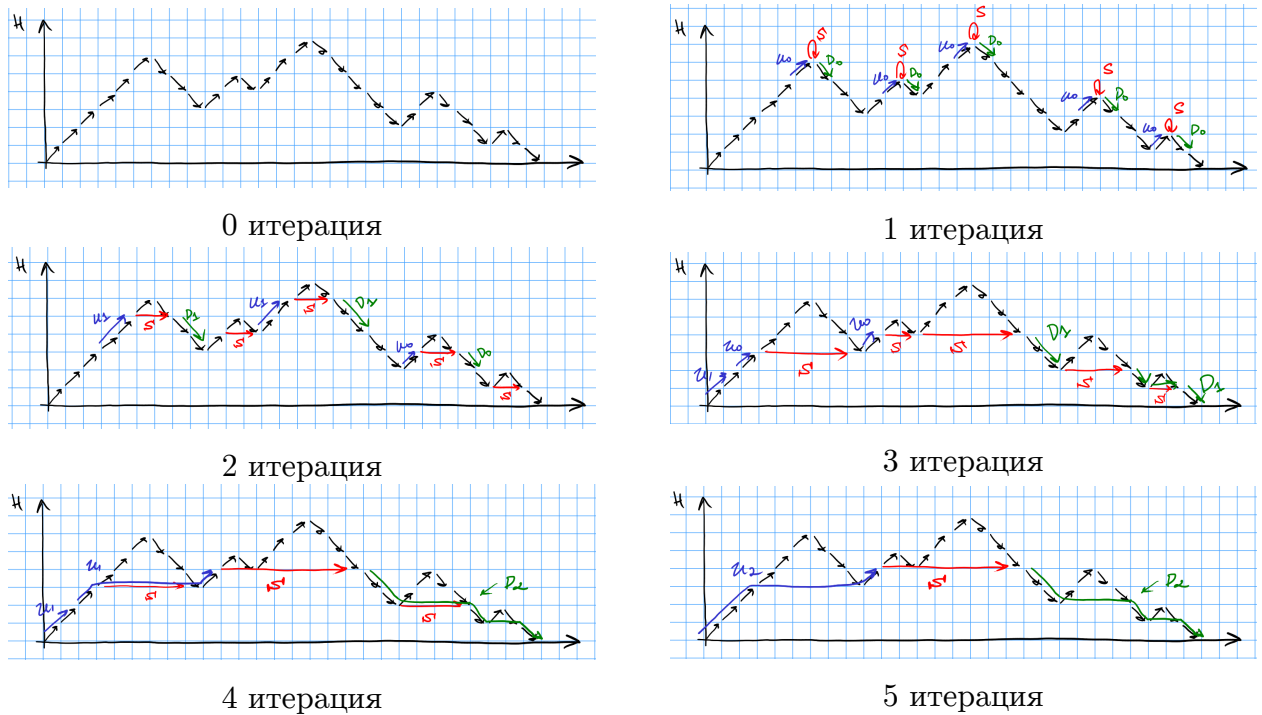


Рис. 5: Пример работы алгоритма (на конкретном пути)

TODO: норм картиночка

4.2.2. Корректность алгоритма

Посмотрим на нетерминал S (опр. 6). В нём присутствуют все продукции из стандартного вида грамматики для языка \mathcal{D}_1 , так что нужно доказать лишь, что все строки, которые выводятся остальными продукциями тоже являются корректными.

Индукцией по длине строки несложно доказать, что:

1. U_i выводит корректные (с балансом на всей строке ≥ 0) строки с балансом 2^i
2. D_i выводит корректные строки с балансом -2^i
3. S выводит корректные строки из \mathcal{D}_1 (с балансом 0)

База очевидна, переход тоже.

4.2.3. Время работы

Теорема 4.1. *Время работы алгоритма 4 на языке Дика на одном типе скобок — $\mathcal{O}(|V|^\omega \log^3 |V|)$.*

Доказательство. Время работы алгоритма П состоит из двух множителей: времени работы одной итерации и числа итераций.

Время работы одной итерации в алгоритме 4 составляло $\mathcal{O}(N^\omega)$ (где N — число состояний РКА-произведения), т.к. на каждой итерации считалось транзитивное замыкание. Заметим, что на самом деле достаточно находить транзитивное замыкание отдельно в каждой компоненте РКА (т.к. разные компоненты вообще не связны). Т.к. компоненты РКА грамматики константны, то размер компонент РКА-произведения — $\mathcal{O}(|V|)$. Так что суммарное время работы одной итерации составит $\mathcal{O}(|V|^\omega K)$.

Оценим теперь число итераций. Рассмотрим конкретный путь (= строку) и посчитаем, сколько итераций потребуется, чтобы его сжать.

За $K + \mathcal{O}(1)$ итераций сожмутся все горы исходной строки: длина горы раскладывается на сумму степеней двойки, после i -го шага будут найдены все строго восходящие U_i и D_i , так что на $(i + 1)$ -ом кусок горы длины 2^i сожмётся в одно S -ребро (так как все предыдущие степени уже сжались).

TODO: картинка для этого кейса

После того, как сжались горы исходной строки, начнут сжиматься горы, которые не могли сделать этого раньше. Почему не могли? Потому что сверху на горе было как минимум две горы поменьше, каждая из которых на прошлой итерации сжалась в S -ребро, а потом правило $S \rightarrow S^*$ сжало их в одно большое ребро.

TODO: и для этого

То есть после каждой такой фазы сжимания число гор уменьшается хотя бы в 2 раза. Т.к. изначально гор было не более $\mathcal{O}(|V|^2)$, таких фаз будет $\mathcal{O}(\log |V|)$.

Итого, получаем $\mathcal{O}(\log |V|)$ фаз сжимания гор, каждая из которых работает за $\mathcal{O}(\log |V|)$ итераций алгоритма 4, время работы которого $\mathcal{O}(|V| \log |V|)$, так и получаем заявленную асимптотику.

На самом деле, в большинстве случаев алгоритм отработает быстрее, так как разные горы могут сжиматься независимо и сжатие вертикальных U_i и D_i может также накладываться на предыдущую фазу.

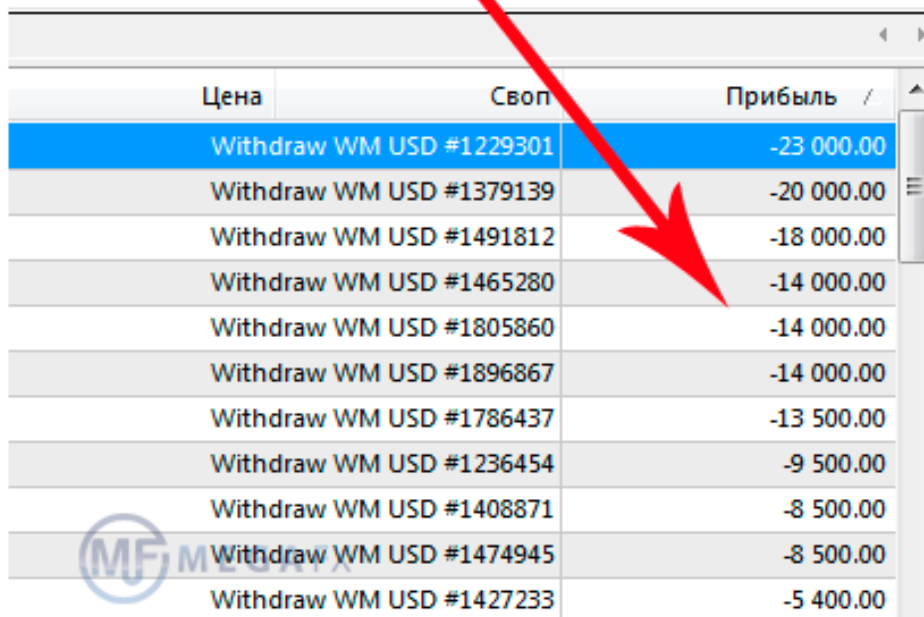
□

Замечание. Этот результат, скорее всего, не обобщается на языки Дика с большим типом скобок.

Мотивация примерно такая: сейчас мы выиграли засчёт того, что для описания состояния автомата нужно примерно одно число (баланс), но для большего типа скобок нужен уже весь стек.

4.3. Выводы и результаты по главе

*Выводы прибыли с одного
из Форекс-счетов*



Цена	Своп	Прибыль /
Withdraw WM USD #1229301		-23 000.00
Withdraw WM USD #1379139		-20 000.00
Withdraw WM USD #1491812		-18 000.00
Withdraw WM USD #1465280		-14 000.00
Withdraw WM USD #1805860		-14 000.00
Withdraw WM USD #1896867		-14 000.00
Withdraw WM USD #1786437		-13 500.00
Withdraw WM USD #1236454		-9 500.00
Withdraw WM USD #1408871		-8 500.00
Withdraw WM USD #1474945		-8 500.00
Withdraw WM USD #1427233		-5 400.00

TODO

5. Результаты (3)

5.1. Алгоритм для смешанного языка Дика

Определение 5.1. TODO: нормально определение в секции с определениями

Пусть у нас есть два языка Дика D_i и D_j над разными алфавитами (например, в одном — $(,)_i$, а в другом $[,]_i$). Тогда в смешанном языке Дика $D_i \odot D_j$ лежат слова, такие что, если рассмотреть проекции на алфавиты D_i и D_j , то это корректные слова из D_i и D_j (*Ужасно..*)

Пример: “ $([()] [()])$ ” — корректное слово из $D_1 \odot D_1$.

Замечание. $D_i \odot D_j$ — это на самом деле пересечение двух КС языков (как обычные языки Дика, но там где ϵ , ещё можно скобки второго типа **TODO**).

Однако КС-языки не замкнуты относительно пересечения, более того, проверять непустоту пересечения двух КС-языков undecidable (**TODO**: link)

Лемма 5.1. [31] В двунаправленном графе, если между парой вершин (u, v) существует какой-то $D_1 \odot D_1$ путь, то существует и такой путь, на котором в любой момент времени вложенность хотя бы одного типа скобок не превышает $6n$.

В доказательстве крутят циклы, в подробности я не вдавалась

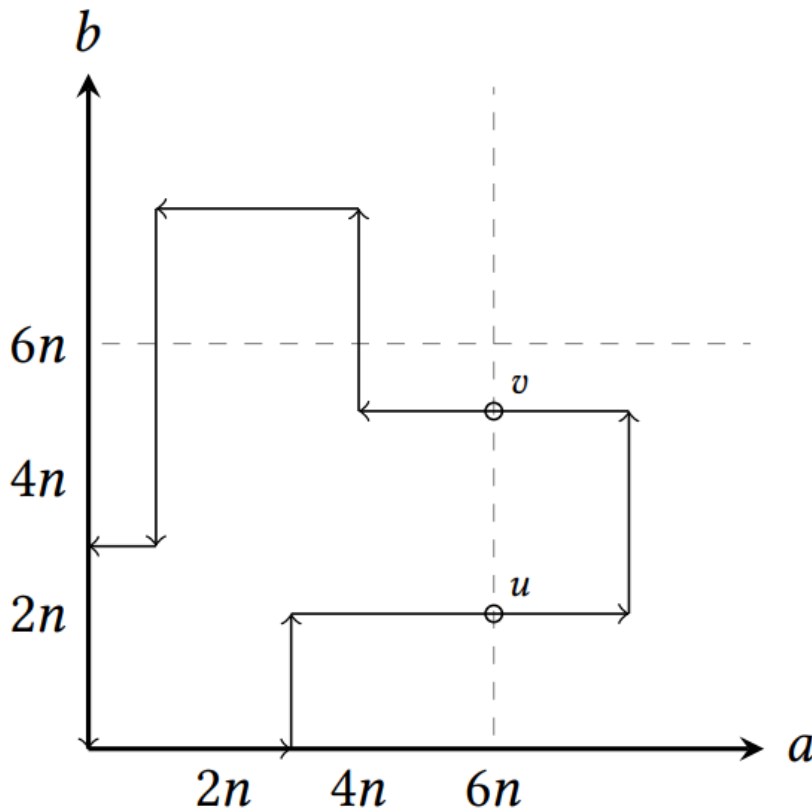


Рис. 6: Путь (кродеться)

Замечание. То есть путь можно искать следующего вида: он в основном ходит внутри квадрата $[0, 6n] \times [0, 6n]$, иногда вылезая из него, но только по одной координате.

Т.е. нужно разобрать отдельно две сущности: куски путей внутри квадрата, и куски путей, которые выходят погулять.

Вооружившись этим знанием, соорудим вот какой алгоритм:

1. Часть про пути в квадратике

Помним (из прошлой главы), что если хотим решать D_1 -достижимость, зная, что вложенность ограничена, можем это с помощью ДКА делать.

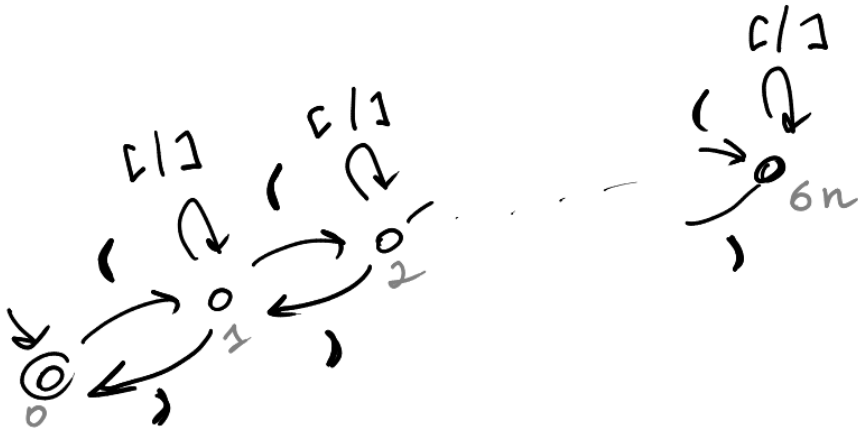


Рис. 7: ДКА для круглых скобок

Строим такие ДКА для обоих типов скобок. Т.к. и тот, и другой регулярный, то можем их прекрасно пересечь с нашим графом.

Получается такой граф троек (u, b, p) (вершина, баланс квадратных скобок, баланс круглых скобок) на $\mathcal{O}(n^3)$ вершинах и $\mathcal{O}(n^2m)$ рёбрах.

Ещё в нём будут ε -рёбра, найденные второй частью алгоритма. Эти рёбра имеют вид $(u, 6n, a) \rightarrow (v, 6n, b)$. Для всех возможных пар a/b и u/v их $\mathcal{O}(n^4)$.

Т.к. входной граф двунаправленный, то полученное произведение будет неориентированным (*это не совсем тривиально, но понятно*), значит достижимость на нём можно искать просто dfs'ом за $\mathcal{O}(n^4)$.

2. Часть про гуляющие пути

В какой момент пути идут гулять? Когда вложенность одной из скобок равно $6n$. А когда возвращаются? Тоже когда равно $6n$. А во все моменты между $\geq 6n$.

То есть хотим искать такие пути, что по одной скобке они просто сбалансированы ($6n \rightsquigarrow 6n$), а по другой какие попало, но всё время не больше $6n$ ($a \rightsquigarrow b$, $a, b \leq 6n$).

Опять-таки, решаем через пересечение языков. Первый — наш граф (L_1) . Вторым — просто язык Дика (L_2) (опять-таки с оговорками про то, что скобочка другого типа — это то же, что и ε). Третий — пути Дика, начинающиеся в балансе a и заканчивающиеся в балансе b (L_3) (решаем для всех возможных пар балансов, чтобы рёбра провести). Давайте решать отдельно для каждого a , то есть нам нужен язык Диковых последовательностей со стартовым балансом a , таких, что их баланс не уходит в минус. Язык, опять-таки регулярный.

Дальше пересекаем. Сначала пересекаем $L_1 \cap L_2$. Это просто прямое произведение — граф пар (u, b) (для каждого a отдельно, снаружи всего это как бы фор по a). Вершин в нём $\mathcal{O}(n^2)$, рёбер $\mathcal{O}(nm)$. Теперь нужно это ещё пересечь с L_3 . Ну так это же просто язык Дика, а Дикову достижимость мы за $\mathcal{O}(m^*\alpha)$ на двунаправленных графах решаем (а произведение $L_1 \cap L_2$ как раз двунаправленное). Получаем $\mathcal{O}(n \cdot nm^*\alpha) = \mathcal{O}(n^4\alpha)$.

5.2. Выводы и результаты по главе

Заключение



TODO

Список литературы

- [1] Abiteboul Serge, Hull Richard, Vianu Victor. Foundations of Databases. — Addison-Wesley, 1995. — ISBN: 0-201-53771-0.
- [2] Aho Alfred V., Hopcroft John E. The Design and Analysis of Computer Algorithms. — 1st edition. — USA : Addison-Wesley Longman Publishing Co., Inc., 1974. — ISBN: 0201000296.
- [3] Aho Alfred V, Hopcroft John E, Ullman Jeffrey D. Time and tape complexity of pushdown automaton languages // Information and Control. — 1968. — Vol. 13, no. 3. — P. 186–206.
- [4] Alman Josh, Williams Virginia Vassilevska. A Refined Laser Method and Faster Matrix Multiplication. — 2020. — 2010.05846.
- [5] Alon Noga, Galil Zvi, Margalit Oded. On the exponent of the all pairs shortest path problem // Journal of Computer and System Sciences. — 1997. — Vol. 54, no. 2. — P. 255–262.
- [6] Analysis of Recursive State Machines / Rajeev Alur, Michael Benedikt, Kousha Etessami et al. — 2005. — Jul. — Vol. 27, no. 4. — P. 786–818. — Access mode: <https://doi.org/10.1145/1075382.1075387>.
- [7] Barrett Chris, Jacob Riko, Marathe Madhav. Formal-Language-Constrained Path Problems // SIAM J. Comput. — 2000. — 01. — Vol. 30. — P. 809–837.
- [8] Bradford Phillip G. Efficient exact paths for Dyck and semi-Dyck labeled path reachability // 2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON) / IEEE. — 2017. — P. 247–253.
- [9] Ceri Stefano, Gottlob Georg, Tanca Letizia. What you Always Wanted to Know About Datalog (And Never Dared to Ask). // Knowledge and Data Engineering, IEEE Transactions on. — 1989. — 04. — Vol. 1. — P. 146 – 166.
- [10] Chatterjee Krishnendu, Choudhary Bhavya, Pavlogiannis Andreas. Optimal Dyck Reachability for Data-Dependence and Alias Analysis // Proc. ACM Program. Lang. — 2017. — Dec. — Vol. 2, no. POPL. — Access mode: <https://doi.org/10.1145/3158118>.
- [11] Chaudhary Anoop, Faisal Abdul. Role of graph databases in social networks. — 2016. — 06.
- [12] CFL-reachability in subcubic time : Rep. / Technical report, IBM Research Report RC24126 ; Executor: Swarat Chaudhuri : 2006.

- [13] Chaudhuri Swarat. Subcubic Algorithms for Recursive State Machines. — POPL '08. — New York, NY, USA : Association for Computing Machinery, 2008. — P. 159–169. — Access mode: <https://doi.org/10.1145/1328438.1328460>.
- [14] Chistikov Dmitry, Majumdar Rupak, Schepper Philipp. Subcubic Certificates for CFL Reachability // arXiv preprint arXiv:2102.13095. — 2021.
- [15] Chomsky Noam. On certain formal properties of grammars // Information and Control. — 1959. — Vol. 2, no. 2. — P. 137–167. — Access mode: <https://www.sciencedirect.com/science/article/pii/S0019995859903626>.
- [16] Context-Free Path Querying by Kronecker Product / Egor Orachev, Ilya Epelbaum, Rustam Azimov, Semyon Grigorev // Advances in Databases and Information Systems / Ed. by Jérôme Darmont, Boris Novikov, Robert Wrembel. — Cham : Springer International Publishing, 2020. — P. 49–59.
- [17] Deleage Jean-Luc, Pierre Laurent. The Rational Index of the Dyck Language $D_1'^*$ // Theor. Comput. Sci. — 1986. — Nov. — Vol. 47, no. 3. — P. 335–343.
- [18] Donaghey Robert, Shapiro Louis W. Motzkin numbers // Journal of Combinatorial Theory, Series A. — 1977. — Vol. 23, no. 3. — P. 291–301. — Access mode: <https://www.sciencedirect.com/science/article/pii/0097316577900206>.
- [19] An Experimental Study of Context-Free Path Query Evaluation Methods / Jochem Kuijpers, George Fletcher, Nikolay Yakovets, Tobias Lindaaker. — SSDBM '19. — New York, NY, USA : Association for Computing Machinery, 2019. — P. 121–132. — Access mode: <https://doi.org/10.1145/3335783.3335791>.
- [20] Fast algorithms for Dyck-CFL-reachability with applications to alias analysis / Qirun Zhang, Michael R Lyu, Hao Yuan, Zhendong Su // Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation. — 2013. — P. 435–446.
- [21] Galler Bernard A., Fisher Michael J. An Improved Equivalence Algorithm // Commun. ACM. — 1964. — May. — Vol. 7, no. 5. — P. 301–303. — Access mode: <https://doi.org/10.1145/364099.364331>.
- [22] Heintze N., McAllester D. On the cubic bottleneck in subtyping and flow analysis // Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science. — 1997. — P. 342–351.
- [23] Hellings Jelle. Path Results for Context-free Grammar Queries on Graphs // CoRR. — 2015. — Vol. abs/1502.02242. — 1502.02242.

- [24] Hellings Jelle. Querying for Paths in Graphs using Context-Free Path Queries. — 2016. — 1502.02242.
- [25] Hopcroft John E., Ullman Jeffrey D. Set merging algorithms // SIAM Journal on Computing. — 1973. — Vol. 2, no. 4. — P. 294–303.
- [26] Hopcroft John E, Ullman Jeffrey D. An introduction to automata theory, languages, and computation. — Upper Saddle River, NJ : Pearson, 1979.
- [27] Ibaraki T., Katoh N. On-line computation of transitive closures of graphs // Information Processing Letters. — 1983. — Vol. 16, no. 2. — P. 95–97. — Access mode: <https://www.sciencedirect.com/science/article/pii/0020019083900339>.
- [28] Impagliazzo Russell, Paturi Ramamohan. On the complexity of k -SAT // Journal of Computer and System Sciences. — 2001. — Vol. 62, no. 2. — P. 367–375.
- [29] Kahlon Vineet. Boundedness vs. unboundedness of lock chains: Characterizing decidability of pairwise cfl-reachability for threads communicating via locks // 2009 24th Annual IEEE Symposium on Logic In Computer Science / IEEE. — 2009. — P. 27–36.
- [30] Kodumal John, Aiken Alex. The Set Constraint/CFL Reachability Connection in Practice. — PLDI '04. — New York, NY, USA : Association for Computing Machinery, 2004. — P. 207–218. — Access mode: <https://doi.org/10.1145/996841.996867>.
- [31] Li Yuanbo, Zhang Qirun, Reps Thomas. On the Complexity of Bidirected Interleaved Dyck-Reachability // Proc. ACM Program. Lang. — 2021. — Jan. — Vol. 5, no. POPL. — Access mode: <https://doi.org/10.1145/3434340>.
- [32] Mathiasen Anders Alnor, Pavlogiannis Andreas. The Fine-Grained and Parallel Complexity of Andersen's Pointer Analysis // Proc. ACM Program. Lang. — 2021. — Jan. — Vol. 5, no. POPL. — Access mode: <https://doi.org/10.1145/3434315>.
- [33] Medeiros Ciro M., Musicante Martin A., Costa Umberto S. Efficient Evaluation of Context-Free Path Queries for Graph Databases // Proceedings of the 33rd Annual ACM Symposium on Applied Computing. — SAC '18. — New York, NY, USA : Association for Computing Machinery, 2018. — P. 1230–1237. — Access mode: <https://doi.org/10.1145/3167132.3167265>.
- [34] Melski David, Reps Thomas. Interconvertibility of a class of set constraints and context-free-language reachability // Theoretical Computer Science. — 2000. — Vol. 248, no. 1. — P. 29–98. — PEPM'97. Access mode: <https://www.sciencedirect.com/science/article/pii/S0304397500000499>.

- [35] Mendelzon Alberto O., Wood Peter T. Finding Regular Simple Paths in Graph Databases // SIAM Journal on Computing. — 1995. — Vol. 24, no. 6. — P. 1235–1258. — <https://doi.org/10.1137/S009753979122370X>.
- [36] Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility / Marco L Carmosino, Jiawei Gao, Russell Impagliazzo et al. // Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science. — 2016. — P. 261–270.
- [37] On economical construction of the transitive closure of an oriented graph / Vladimir L’vovich Arlazarov, Yefim A Dinitz, MA Kronrod, IgorAleksandrovich Faradzhhev // Doklady Akademii Nauk / Russian Academy of Sciences. — 1970.
- [38] One Algorithm to Evaluate Them All: Unified Linear Algebra Based Approach to Evaluate Both Regular and Context-Free Path Queries / E. Shemetova, Rustam Azimov, Egor Orachev et al. // ArXiv. — 2021. — Vol. abs/2103.14688.
- [39] RDF - Semantic Web Standards. — Accessed: 2020-05-15. Access mode: <https://www.w3.org/RDF/>.
- [40] Rehof Jakob, Fähndrich Manuel. Type-Base Flow Analysis: From Polymorphic Subtyping to CFL-Reachability // Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. — POPL ’01. — New York, NY, USA : Association for Computing Machinery, 2001. — P. 54–66. — Access mode: <https://doi.org/10.1145/360204.360208>.
- [41] Reps Thomas. Program analysis via graph reachability // Information and Software Technology. — 1998. — Vol. 40, no. 11. — P. 701–726. — Access mode: <https://www.sciencedirect.com/science/article/pii/S0950584998000937>.
- [42] Reps Thomas. Undecidability of context-sensitive data-dependence analysis // ACM Transactions on Programming Languages and Systems (TOPLAS). — 2000. — Vol. 22, no. 1. — P. 162–186.
- [43] SPARQL 1.1 overview. — Accessed: 2020-05-15. Access mode: <https://www.w3.org/TR/sparql11-query/>.
- [44] Santos Fred C., Costa Umberto S., Musicante Martin A. A Bottom-Up Algorithm for Answering Context-Free Path Queries in Graph Databases // Web Engineering / Ed. by Tommi Mikkonen, Ralf Klamma, Juan Hernández. — Cham : Springer International Publishing, 2018. — P. 225–233.

- [45] Scott Elizabeth, Johnstone Adrian. GLL Parsing // Electronic Notes in Theoretical Computer Science. — 2010. — Vol. 253, no. 7. — P. 177–189. — Proceedings of the Ninth Workshop on Language Descriptions Tools and Applications (LDTA 2009). Access mode: <https://www.sciencedirect.com/science/article/pii/S1571066110001209>.
- [46] Scott Elizabeth, Johnstone Adrian, Hussain Shamsa Sadaf. Tomita-style generalised LR parsers // Royal Holloway University of London. — 2000.
- [47] Sevon Petteri, Eronen Lauri. Subgraph Queries by Context-free Grammars // Journal of Integrative Bioinformatics. — 2008. — Vol. 5, no. 2. — P. 157–172. — Access mode: <https://doi.org/10.1515/jib-2008-100>.
- [48] Summary-based context-sensitive data-dependence analysis in presence of callbacks / Hao Tang, Xiaoyin Wang, Lingming Zhang et al. // Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. — 2015. — P. 83–95.
- [49] Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture / Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, Thatchaphol Saranurak // Proceedings of the forty-seventh annual ACM symposium on Theory of computing. — 2015. — P. 21–30.
- [50] 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.
- [51] Wikipedia contributors. Datalog — Wikipedia, The Free Encyclopedia. — 2021. — [Online; accessed 6-May-2021]. Access mode: <https://en.wikipedia.org/w/index.php?title=Datalog&oldid=1015453010>.
- [52] Williams Virginia Vassilevska. On some fine-grained questions in algorithms and complexity // Proceedings of the ICM / World Scientific. — Vol. 3. — 2018. — P. 3431–3472.
- [53] Yan Dacong, Xu Guoqing, Rountev Atanas. Demand-driven context-sensitive alias analysis for Java // Proceedings of the 2011 International Symposium on Software Testing and Analysis. — 2011. — P. 155–165.
- [54] Yannakakis Mihalis. Graph-Theoretic Methods in Database Theory // Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. — PODS '90. — New York, NY, USA : Association for Computing Machinery, 1990. — P. 230–242. — Access mode: <https://doi.org/10.1145/298514.298576>.

- [55] Yannakakis Mihalis. Hierarchical state machines // IFIP International Conference on Theoretical Computer Science / Springer. — 2000. — P. 315–330.
- [56] Yellin Daniel M. Speeding up Dynamic Transitive Closure for Bounded Degree Graphs // Acta Inf. — 1993. — Apr. — Vol. 30, no. 4. — P. 369–384. — Access mode: <https://doi.org/10.1007/BF01209711>.
- [57] Younger Daniel H. Recognition and parsing of context-free languages in time n^3 // Information and Control. — 1967. — Vol. 10, no. 2. — P. 189–208. — Access mode: <https://www.sciencedirect.com/science/article/pii/S001999586780007X>.
- [58] 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.
- [59] Yuval Gideon et al. AN ALGORITHM FOR FINDING ALL SHORTEST PATHS USING $N^{2.81}$ INFINITE-PRECISION MULTIPLICATIONS. — 1976.
- [60] Zarrinkalam Fattane, Kahani Mohsen, Paydar Samad. Using graph database for file recommendation in PAD social network // 7'th International Symposium on Telecommunications (IST'2014). — 2014. — P. 470–475.
- [61] Zhang Qirun. Conditional Lower Bound for Inclusion-Based Points-to Analysis // arXiv preprint arXiv:2007.05569. — 2020.
- [62] Zheng Xin, Rugina Radu. Demand-driven alias analysis for C // Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. — 2008. — P. 197–208.