Оглавление

Введение

Актуальность

Графовые модели данных широко используются в различных областях науки, например, в биоинформатике [57], анализе социальных сетей [73, 13], графовых базах данных [38, 67] и разных видах статического анализа [50].

Важной задачей 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 [40]. 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, CFPQ (Context-Free Path Querying) была впервые сформулирована в терминах запросов к графовым базам данных [67], 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 [39, 26, 54], в большинстве своём основанных на различных методах синтаксического анализа.

К сожалению, все существующие решения задачи в общем случае недостаточно эффективны для использования на практике [21]. Более того, существует условная нижная оценка [24], 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 двунаправленных графах [71, 12]. 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.

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.

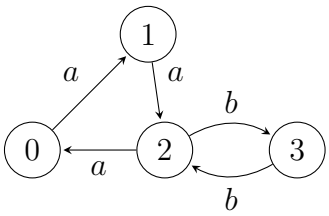


Рис. 1: Помеченный граф с $\Sigma = \{a, b\}$

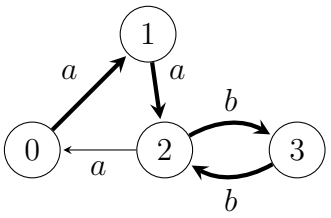


Рис. 2: На пути $v_0 \to v_1 \to v_2 \to v_3 \to v_2$ читается слово $aabb$

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

Говорят, что слово u *выводится из слова* w *за один шаг* ($w \Rightarrow u$), где $u, w \in (\Sigma \cup N)^*$, если $w = \alpha\, B\, \gamma, u = \alpha\, \beta\, \gamma$, и $B \to \beta \in P$, то есть u получается из w применением продукции к нетерминалу B.

Слово u *выводится* из слова w ($w \Rightarrow^* u$), если $\exists w_0, w_1, \ldots, w_n$, где $w_0 = w, w_n = u$ и $\forall 0 \leqslant i < n$ верно, что $w_i \Rightarrow w_{i+1}$ (то есть, существует последовательность выводов за один шаг, приводящая w к u).

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

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

Пример 0.1. *Опишем грамматику* $\mathcal{G} = \langle \Sigma, N, S, P \rangle$*, задающую язык* $L(\mathcal{G})$ *слов вида* $b^n a^m b^{2n}$ *для* $n, m \geqslant 0$.

Алфавит состоит из букв a *и* b: $\Sigma = \{a, b\}$*, нетерминалов будет два:* S *– стартовый нетерминал и* A *— нетерминал для слов вида* a^m*. Также для вывода слов в грамматике будут использоваться следующие продукции:*

$$S \to bSbb$$
$$S \to A$$
$$A \to aA$$
$$A \to \varepsilon$$

где ε — пустое слово.

Продукции для одного нетерминала иногда пишут через | (логическое или), так что короткая запись грамматики для L выглядит так:

$$L\colon S \to bSbb \mid A$$
$$A \to aA \mid \varepsilon$$

Покажем, как в такой грамматике вывести слово 'bb aaa bbbb':
$S \Rightarrow \boldsymbol{bSbb} \Rightarrow b\boldsymbol{bSbb}bb \Rightarrow bb\boldsymbol{A}bbbb \Rightarrow bb\boldsymbol{aA}bbbb \Rightarrow bba\boldsymbol{aA}bbbb \Rightarrow bbaa\boldsymbol{aA}bbbb \Rightarrow bb\,aaa\,bbbb$

Теперь определим саму задачу.

Определение 0.4. Пусть даны ориентированный помеченный граф G и контекстно-свободная грамматика $\mathcal{G}$ над тем же алфавитом (то есть алфавиты меток графа и терминалов грамматики совпадают). Тогда *задача поиска путей с контекстно-свободными ограничениями (сокращённо CFPQ)* в реляционной семантике запроса [27] заключается в нахождении всех пар вершин $u, v \in V(G)$, таких что существует путь $p\colon u \rightsquigarrow v$, на котором читается слово $w \in L(\mathcal{G})$.

Пример 0.2. *На рис. 3 и 4 показаны корректный и некорректный пути для языка L слов вида $a^n b^n$ для $n \geqslant 0$, задаваемого грамматикой $L\colon S \to aSb \mid \varepsilon$.*

Корректным также является и непростой путь $v_2 \to v_0 \to v_1 \to v_2 \to v_0 \to v_1 \to v_3 \to v_2 \to v_3 \to v_2 \to v_3 \to v_2$, на котором читается слово aaaaaa bbbbbb (a^6b^6).

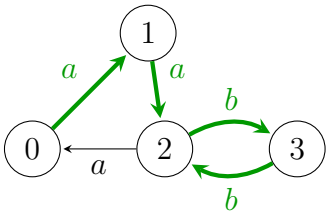


Рис. 3: Корректный путь $v_0 \rightsquigarrow v_2$

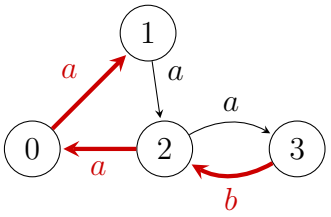


Рис. 4: Некорректный путь $v_3 \rightsquigarrow v_1$

Цель и задачи

Целью работы является получение решений для частных случаев задачи поиска путей с контекстно-свободными ограничениями, основанных 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 и двунаправленные графы.
  - Язык Дика на одном типе скобок.
- Применить выбранный подход для решения задачи достижимости для смешанного языка Дика и двунаправленных графов.

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 была сформулирована Михалисом Яннакакисом ещё в 1990 году [67] в применении к запросам к базам данных, написанных на декларативном языке Datalog [63, 11].

В последнее десятилетие интерес к задаче возродился в контексте запросов к RDF* [48] — графовой модели представления данных в сети, разработанной W3C†. 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 [52]. Однако SPARQL позволяет осуществлять только запросы, представленные в виде регулярных выражений, тогда как некоторые интересные запросы (такие как *same-generation queries*‡ [1]) могут быть выражены только в терминах контекстно-свободных языков.

Задача также нашла широкое применение в разных видах статического анализа [50] (в этой области она более известная под именем задачи контекстно-свободной достижимости или CFL-Reachability), таких как межпроцедурный слайсинг (interprocedural slicing) [58], анализ указателей (points-to analysis) [59, 65], анализ псевдонимов (alias analysis) [75, 66, 22], межпроцедурный анализ потока данных (interprocedural dataflow analysis) [49, 47], анализ формы (shape analysis) [50, 42], свёртка констант (constant propagation) [53] и другие [47, 41, 8].

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 [39] использует тот же подход, что и алгоритм Кока-Янгера-Касами [70] парсинга контекстно-свободных языков: приведение грамматики к нормальной форме Хомского [17] и использование метода 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 [14] улучшил этот алгоритм, уменьшив асимптотику в $\mathcal{O}(\log|V|)$ раз, используя метод четырёх русских [45].

Алгоритм Григорьева и Рогозиной, основан на обобщённом нисходящем синтаксическом анализе§ — GLL [55] парсинге, и работает за $\mathcal{O}(|V|^3 \max_{v \in V}(deg^+(v)))$. Алгоритм Медейроса и др. [38] также основан на нисходящем синтаксическом анализе — LL парсинге, но имеет время работы $\mathcal{O}(|V|^3|P|)$, где $|P|$ — число продукций входной

*Resource Description Framework — Среда описания ресурса

†World Wide Web Consortium — Консорциум Всемирной паутины

‡Запросы поиска объектов, находящихся на одном уровне иерархии

§Top-down parsing

грамматики.

Алгоритм Сантоса и др. [54] основан на восходящем синтаксическом анализе* — Tomita-Style Generalized LR парсинге [56], однако его время работы не оценено, 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 [54] сводят задачу CFPQ к транзитивному замыканию матриц (по аналогии с решением Валианта [62] задачи распознавания контекстно-свободных языков). Преимущество этого алгоритма в том, что он использует операции над матрицами, которые могут быть соптимизированы с использованием GPGPU†.

Хеллингс в своей работе [26] рассматривает задачу в основанной на путях (path-based) семантике запроса и разрешает её, используя аннотированные грамматики.

Чаудхури [15], а также Орачев и др. [18] 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 [39, 14]) может быть соптимизирован [46] в $\mathcal{O}(\log|NV|)$ 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. [21], реализовав и замерив производительность трёх алгоритмов: Хеллингса [26], Сантоса и др. [54] и Азимова и др. [54].

Более того, скорее всего, решения с более быстрой ($\mathcal{O}(|V|^{3-\varepsilon})$) асимптотикой не существует. Это так называемый "cubic bottleneck"‡ [24] данной задачи. Было доказано, что она является 2NPDA§-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 [12] построили условную нижнюю оценку, сведя к задаче CFPQ (а именно, к $\mathcal{D}_k$-достижимости (опр. 1.2)) задачу BMM¶, на которую есть условная нижняя оценка. А именно, согласно BMM-гипотезе [64], не суще-

*Bottom-up parsing

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

‡Узкое место

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

¶Boolean Matrix Multiplication — перемножение двух булевых матриц

ствует субкубического *комбинаторного*[*] алгоритма для перемножения двух булевых матриц. Замечание про комбинаторность алгоритма важно, так как алгебраическое субкубическое решения для BMM существует, а именно, она сводится к обычному перемножению матриц, которое может быть совершено за $\mathcal{O}(|V|^\omega)$[†].

Позднее Чжан [74] улучшил эту оценку, построив сведение BMM к $\mathcal{D}_1$-достижимости, тем самым 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 ситуация не так хороша. Нижняя оценка в $\mathcal{O}(|V|^\omega)$ вытекает из сведения от BMM, 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. [16] показали, что при условии NSETH[‡] [43] не существует нижней оценки, основанной на SETH[§] [31] 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 контекстно-свободная грамматика, а иногда ещё и известны 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 контекстно-свободного распознавания (context-free recognition), так как граф-цепочка — это просто одна строка. А для задачи контекстно-свободного распознавания существует субкубическое решение [62], более того, она эквивалентна задаче перемножения булевых матриц (так что решается за $\mathcal{O}(|V|^\omega)$). В своей работе [67] Яннакакис также заметил, что это сведение можно обобщить на случай ацикличных графов.

Решение с такой же асимптотикой $\mathcal{O}(|V|^\omega)$ получено для ещё одного частного случая — иерархических грамматик[¶] [68]. На самом деле такие грамматики задают регулярные языки, однако размер автомата может быть экпоненциален относительно

[*]Этот термин не вполне определен, но можно понимать его как “не алгебраический”. В частности, комбинаторные алгоритмы не должны использовать деление и вычитание, так как те пользуются особенностями алгебраических структур (а именно, существованием обратного)

[†]$2 < \omega < 2.373$ [4]

[‡]Nondeterministic Strong Exponential Time Hypothesis — Недетерминированная сильная гипотеза об экспоненциальном времени

[§]Strong exponential time hypothesis — Сильная гипотеза об экспоненциальном времени

[¶]Hierarchical state machines

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 [67] к построению транзитивного замыкания входного графа (опр. 2.4).

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

Определение 1.1. *Языком Дика*[*] 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 (или ПСП) на k типах скобок.

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

Определение 1.2. Задачу CFPQ для языка Дика называют также *задачей Диковой достижимости*[†] [34] 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 [60], обращение по указателю и их разыменование [75], запись и чтение из поля [66], взятие и возврат блокировки [32].

Первыми улучшениями для задачи Диковой достижимости стали лучшие оценки на время работы уже существующих алгоритмов. Так, Кодумал и Айкен [34] показали, что алгоритм Репса [39], 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 [49] 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 потока (type-based flow analysis).

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 [10] задачу $\mathcal{D}_1$-достижимости к $\mathcal{O}(\log^2 |V|)$ перемножениям AGMY-матриц[‡], каждое из которых может быть произведено за время $\mathcal{O}(|V|^\omega \log |V|)$. Матиасен и др. [37] строят "более комбинаторное" решение, сводя задачу $\mathcal{D}_1$-достижимости к $\mathcal{O}(\log^2)$ непосредственно перемножениям 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*.

[*] Dyck language
[†] Dyck-reachability
[‡] Alon, Galil, Margalit [5]; и Yuval [72]

Определение 1.3. Помеченный граф $G = \langle V, E, \Sigma_k \rangle$ называют *двунаправленным* (bidirected graph) [71], 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.

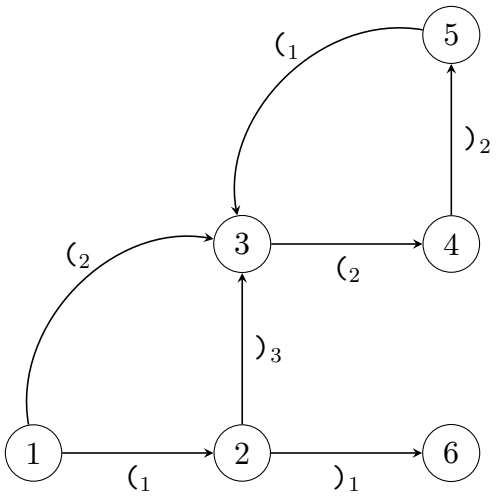


Рис. 5: Обычный ориентированный граф

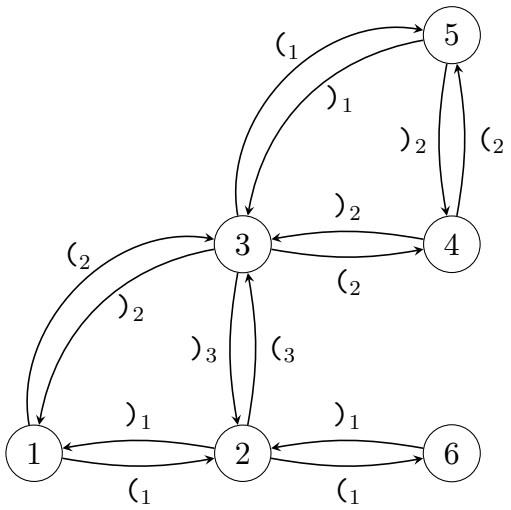


Рис. 6: Двунаправленный граф

В своей работе Юань и др. [71] заметили, что при решении задачи анализа указателей (points-to analysis) граф получается двунаправленным (так как отношения `GetField` и `PutField` взаимообратные). 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-префиксов* его поддерева. Чжан и др. [22], заметив, что в двунаправленных графах Диково достижимые вершины формируют классы эквивалентности (лемма 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. [12] улучшили этот результат до $\mathcal{O}(|E| + |V|\alpha(|V|))$†.

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

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

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

‡Context sensitivity

Определение 1.4. Для двух языков L_1 и L_2, заданных над алфавитами Σ_1 и Σ_2 соответственно, определим *оператор смешения*[*] [36] $\odot : L_1 \times L_2 \to (\Sigma_1 \cup \Sigma_2)^*$ 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:
$L_1 \odot L_2 = \bigcup\limits_{a \in L_1, b \in L_2} a \odot b.$

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*[†].

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.

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 случаем неразрешима [51]. Более того, она неразрешима уже для языка $\mathcal{D}_2 \odot \mathcal{D}_2$ даже на двунаправленных графах. Поэтому Ли и др. [36] 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

Задача поиска путей с контекстно-свободными ограничениями была сформулирована более 30 лет назад и применяется в работе с графовыми базами данных и в статическом анализе. Существует множество алгоритмов, для решения этой задачи, в большинстве своём, основанных на алгоритмах парсинга. Однако существующие алгоритмы недостаточно эффективны, более того, существует условная нижняя оценка в $\mathcal{O}(|V|^3)$, которая достигается большинством этих решений.

Для некоторых частных случаев уже были предложены более быстрые (субкубические) решения, все они были описаны в этой главе. Большая часть частных случаев так или иначе связана с языком Дика, так как в задачах статического анализа языком ограничений часто является именно он.

[*] Interleaving operator
[†] Interleaved Dyck language или InterDyck language

2. Алгоритм, основанный на пересечении языков

В качестве единого подхода к построению частичных решений для задачи CFPQ был выбран алгоритм [15, 18], 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 выбраны начальное и конечные состояния.

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

- Q — конечное множество состояний
- Σ — конечный алфавит
- $\delta \colon Q \times \Sigma \to 2^Q$ — 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 состояние в терминальное хотя бы одним способом.

Недетерминированные конечные автоматы задают *регулярные языки.*

Пример 2.1. *Автомат на рисунке 7 задаёт язык слов, содержащих кратное трём число букв* b *и заканчивающихся на* ab*.*
Опишем его подробнее. Алфавит состоит из двух символов a *и* b*:* $\Sigma = \{a, b\}$*, состояний 5 штук:* $Q = \{0, 1, 2, 3, 4\}$*, при этом* $q_0 = 0$*, а терминальное состояние всего одно:* $T = \{4\}$*. Функция перехода* δ *соответствует рёбрам автомата.*

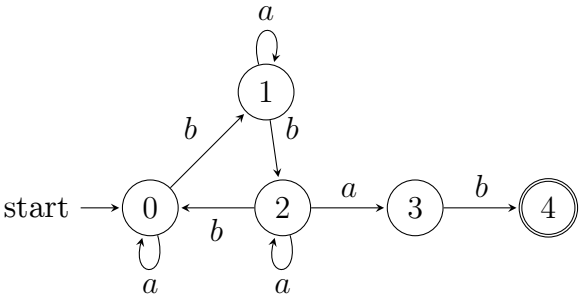


Рис. 7: Пример НКА

[*]Nondeterministic 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.*

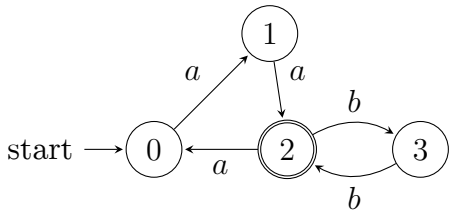


Рис. 8: НКА, задающий слова, читаемые на путях $0 \rightsquigarrow 2$. Например, *aa*, *aabb* или *aabbaaa*

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

Получаем, что для каждой пары вершин u и v, наличие пути $p\colon u \rightsquigarrow v$, 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 какомлибо 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:

- $Q = Q_1 \times Q_2$ (то есть 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. *[29] 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 (или РКА)** [6] — это набор компонент $\langle M_1, M_2, \ldots, M_k\rangle$, 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

*Recursive state machine (или RSM)

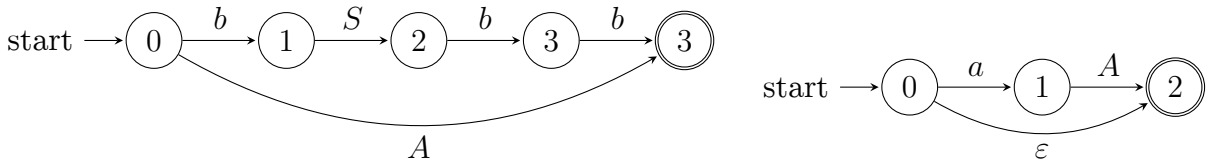
- $Ex_i \subset Q_i$ — множество конечных состояний
- $\delta_i \colon Q_i \times (\Sigma_i \cup \bigcup\limits_{j=1}^{k} En_i \times Ex_i) \to 2^{Q_i}$ — функция перехода. У δ_i есть два типа переходов: *внутренние*, которые работают как обычные переходы в НКА и *рекурсивные*, которые делают вызов другой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

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

Пример 2.2. *Вспомним грамматику для языка слов вида $b^n a^m b^{2n}$ (пример 0.1):*

$$L \colon S \to bSbb \mid A$$
$$A \to aA \mid \varepsilon$$

РКА по грамматике обычно строится следующим образом: создаётся компонента для каждого нетерминала, в которой проводятся пути, соответствующие продукциям.



(a) Компонента S (b) Компонента A

Рис. 9: РКА для языка $b^n a^m b^{2n}$

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 $\mathcal{R} = \langle M_1, \ldots, M_k \rangle, M_i = \langle Q_i, \Sigma_i, En_i, Ex_i, \delta_i \rangle$ 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 (то есть для каждой пары состояний находится, есть ли путь из одного в другое). Так что за один запуск такого алгоритма можно проверить контекстно-свободную достижимость для всех пар вершин исходного графа.

Подводя итог, все алгоритмы для CFPQ, 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 [67] решению задачи достижимости для обычного ориентированного графа и заключается в построении *транзитивного замыкания.*

Определение 2.4. Пусть дан ориентированный граф G. Тогда *транзитивным замыканием графа* G называют такой граф G^+, что $V(G^+) = V(G)$, а ребро $u \to v$ содержится в $E(G^+)$ тогда и только тогда, когда v достижима из u в графе G.

То есть это просто граф, рёбра которого (как бинарное отношение) являются транзитивным замыканием рёбер исходного графа G.

В случае РКА задача достижимости не решается так просто, так как 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. Пусть дан ориентированный граф G. Тогда задача *поддержания инкрементального транзитивного замыкания графа* G заключается в ответах на следующие запросы:

- Провести новое ребро $u \to v$
- Проверить связность вершин u и v

То есть задача заключается в добавлении рёбер (поэтому оно и инкрементальное) и поддержании транзитивного замыкания.

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.

Листинг 1 Алгоритм достижимости для РКА

```
1:  function RSMReachability2(R)
2:      A ← Empty adjacency matrix
3:      Q ← Empty Queue
4:      for i ∈ 1..k do
5:          for u -c-> v ∈ δ_i do
6:              Q.Push(⟨u, v, i⟩)
7:      while Q is not Empty do
8:          ⟨u, v, i⟩ ← Q.Pop()
9:          if u ∈ En_i ∧ v ∈ En_i then                ▷ Нашли новый путь
10:             A ← A ∪ getEdges(i, u, v)
11:             Q.PushAll(getEdges(i, u, v))           ▷ Добавляем новые рёбра
12:         for x ∈ Q_i do
13:             if A_{x,u} ∧ ¬A_{x,v} then
14:                 for y ∈ Q_i do
15:                     if A_{v,y} ∧ ¬A_{x,y} then
16:                         A ← A ∪ ⟨x, y⟩
17:                         Q.Push(⟨x, y, i⟩)          ▷ 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 [30]. 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 проходя через него. А именно, пусть было добавлено ребро $u \to v$. 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 ($v \rightsquigarrow y$).

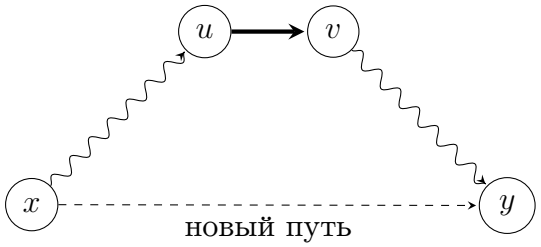


Рис. 10: Нахождение нового пути

Если новое ребро транзитивного замыкания (= 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 $O(|E^*|)$. 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 ($x \to v$), 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 $\mathcal{O}(\log|V|)$ раз [46], воспользовавшись методом четырёх русских [45] (так как работа происходит над булевыми векторами).

2.3. Получение решений для частных случаев

Основой для получения частных решений будут модификации 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 [61] задача OMv*.

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

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 [69] 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*

  Существует несколько алгоритмов (например [33]) для работы с динамически меняющимися планарными графами. Однако очень трудно гарантировать планарность РКА-произведения (тем более, при вставке рёбер), хотя бы потому, например, что планарные графы имеют линейное относительно числа вершин число рёбер.

- *Автоматы с ограниченным стеком* [15]*

  Для автоматов, РКА которых не уходит в рекурсию, был построен [15] алгоритм, с временем работы $\mathcal{O}(|V|^3/\log^2|V|)$.

- *Неориентированные графы*

  Подробнее рассмотрены в разделе 3.1

- *Графы с малым числом итераций добавления рёбер*

  Подробно рассмотрены в разделе 4.1

2.4. Выводы и результаты по главе

В этой главе введены основные определения и идея пересечения языков. Также в ней описан алгоритм 1, основанный на данной идее и реализующий поддержку инкрементального транзитивного замыкания. Решения для частных случаев, представленные в следующих двух главах, являются модификациями этого алгоритма.

*Bounded-stack State Machines

3. Язык Дика и двунаправленные графы

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)** [23] — 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.

Листинг 2 Алгоритм неориентированной достижимости для РКА

```
1:  function UndirectedRSMReachability(R)
2:      A ← Empty adjacency matrix
3:      Q ← Empty Queue
4:      D ← DSU(| ⋃_{i=1}^{k} Q_i |)
5:      for i ∈ 1..k do
6:          for u -c-> v ∈ δ_i do
7:              Q.Push(⟨u, v, i⟩)
8:      while Q is not Empty do
9:          ⟨u, v, i⟩ ← Q.Pop()
10:         if u ∈ En_i ∧ v ∈ En_i then                ▷ Нашли новый путь
11:             A ← A ∪ getEdges(i, u, v)
12:             Q.PushAll(getEdges(i, u, v))
13:             D.Union(u, v)                          ▷ Добавляем новые рёбра
14:     return A
```

*Disjoint-set/union–find 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 → конечное состояние⟩.

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

Листинг 3 Система Непересекающихся Множеств

```
1:  Structure DisjointSets
2:      function DisjointSets(V)
3:          for v ∈ V do
4:              P[v] ← v                                   ▷ Предкок
5:              R[v] ← 0                                   ▷ Ранг
6:          for v ∈ En(V) do
7:              En[v] ← {v}                ▷ Список стартовых вершин поддерева
8:          for v ∈ Ex(V) do
9:              Ex[v] ← {v}                ▷ Список конечных вершин поддерева
10:     function Find(v)
11:         if P[v] = v then return v
            return P[v] = Find(P[v])                      ▷ Эвристика сжатие путей
12:     function Union(u, v)
13:         u ← Find(u)
14:         v ← Find(v)
15:         if u = v then return
16:         if R[u] > R[v] then
17:             Swap(u, v)                                ▷ Ранговая эвристика
18:         Q.PushAll({⟨en_u, ex_v⟩ | en_u ∈ En[u], ex_v ∈ Ex[v]})
19:         Q.PushAll({⟨en_v, ex_u⟩ | en_v ∈ En[v], ex_u ∈ Ex[u]})   ▷ Добавление новых
    рёбер
20:         En[v] ← En[v] ∪ En[u]
21:         Ex[v] ← Ex[v] ∪ Ex[u]
22:         R[v] ← max(R[v], R[u] + 1)
23:         P[u] = v                                      ▷ Объединение компонент
24:
```

За основу взята стандартная реализация [28] 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.

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). □

Замечание. Алгоритм 2 очевидно работает корректно на РКА, все рёбра которого неориентированы. Вспомним, что для решения задачи CFPQ алгоритм достижимости запускается на прямом произведении входного графа и РКА входной грамматики. Так что, чтобы произведение было неориентированным, необходима неориентированность и входного графа и РКА входной грамматики.

И если неориентированные графы являются довольно естественными объектами, то неориентированные РКА задают очень специфичные языки (про языки, задаваемые неориентированными НКА можно почитать в [35]).

Так что в следующем разделе будет показана корректность алгоритма 2 для более реалистичного частного случая.

3.2. 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 отношении Диковой достижимости является отношением эквивалентности.*

Доказательство. Действительно, если инвертировать (заменить все $(_i$ на $)_i$ и наоборот) и развернуть правильную скобочную последовательность, то получится так же правильная скобочная последовательность.
Пример: ПСП: ‘([]())’ → развёрнутая: ‘))()[[(’ → инвертированная: ‘(()[])’.

□

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 не совсем стандартный РКА для языка Дика (изображён на рис. 11). Нестандартность заключается в том, что он содержит несколько (а именно, $k+2$) терминальных состояний, тогда как обычно они объединены в одну вершину.

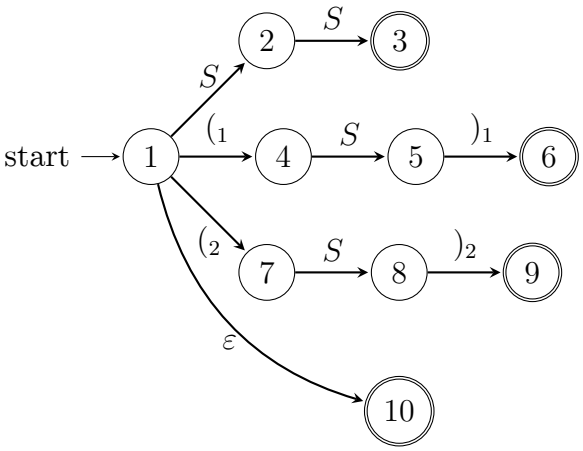


Рис. 11: РКА для языка Дика $\mathcal{D}_2$.
Состоит из одной компоненты S.

Доказательство. Теоремы 3.2

Достаточно доказать, что для любой пары состояний $e \in En_i, ex \in Ex_i$ существование неориентированного пути эквивалентно существованию ориентированного.

$\Leftarrow$ (ориентированный $\Rightarrow$ неориентированный)

Очевидно, если есть ориентированный путь $en \rightsquigarrow ex$, то если убрать ориентацию этот путь всё ещё останется корректным.

$\Rightarrow$ (неориентированный $\Rightarrow$ ориентированный)

Для начала заметим, что компонента нашего РКА (произведения РКА языка Дика и входного графа) образует слоистую структуру: на i-ом слое — состояния $\langle q_i, u\rangle$, где q_i — i-ое состояние РКА языка Дика. Так как РКА Дика топологически отсортирован, все рёбра ведут из слоя с меньшим номером в слой с большим.

Назовём неориентированный путь *простым*, если он проходит по каждому слою не боле одного раза.

Пусть есть неориентированный путь $p\colon en \rightsquigarrow ex$. Покажем, что тогда существует *простой* путь $p'\colon en \rightsquigarrow ex$. Заметим, что так как рёбра идут только из слоя с меньшим номером в слой с большим, то простой путь из первого слоя в последний на самом деле направлен согласно исходной ориентации рёбер. То есть такой простой путь и будет искомым.

Доказывать это утверждение (про наличие простого пути p') будем индукцией по длине пути.

Очевидно, утверждение верно для путей длины 1, 2 и 3 — все они и так являются простыми.

Рассмотрим путь p длины $\geqslant 4$, он уже не будет простым. Найдём первую *точку перегиба* пути — вершину, из который 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 $\langle q_i, u\rangle \to \langle q_j, v\rangle \to \langle q_i, w\rangle$ ($\langle q_j, v\rangle$ — точка перегиба).

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

- открывающая скобка $(_k$

  $p\colon (q_1, u) \to (q_{4/7}, v) \to (q_1, w) \to \cdots \to (q_f, z)$.

  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 $\langle q_1, u\rangle \to \langle q_2, w\rangle$. Также, по предположению индукции существует простой путь $\langle q_1, w\rangle \to \langle q_f, z\rangle$, а значит есть и S-ребро $\langle q_2, w\rangle \to \langle q_3, z\rangle$. Вместе эти два ребра формируют просто путь $\langle q_1, u\rangle \to \langle q_2, w\rangle \to \langle q_3, z\rangle$, что и хотелось.

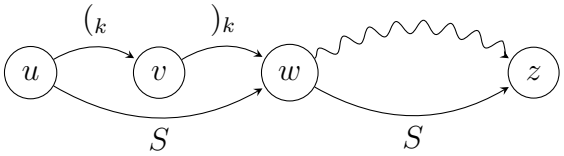


Рис. 12: Случай с меткой $(_k$

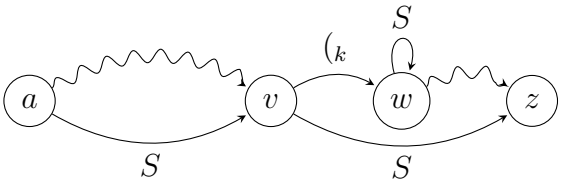


Рис. 13: Случай с меткой $)_k$

- S-метки

  $p\colon (q_1, a) \to \cdots \to (q_i, u) \to (q_j, v) \to (q_i, w) \to \cdots \to (q_f, z)$.

  По лемме 3.1, из наличия Дикового пути $w \rightsquigarrow v$ следует наличие Дикового 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 $S \to SS$, 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 $)_k$

  $p\colon (q_1, a) \to \cdots \to (q_i, u) \to (q_f, v) \to (q_i, w) \to \cdots \to (q'_f, z)$.

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

□

3.3. Выводы и результаты по главе

В этой главе была представлена модификация алгоритма 1, предполагающая неориентированность данного ей РКА. Алгоритм 2 использует структуру данных Система Непересекающихся Множеств для поддержания инкрементального транзитивного замыкания неориентированного графа. Также была доказана корректность данного алгоритма для двунаправленных графов и языка Дика.

4. Язык Дика на одном типе скобок

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 неикрементальном 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.

Листинг 4 Алгоритм достижимости для РКА

```
1:  function RSMReachability(R)
2:      A ← Adjacency matrix for R
3:      while A is changing do
4:          A' ← transitiveClosure(A)              ▷ Построение транзитивного замыкания
5:          for i ∈ 1..k do
6:              for u ∈ En_i do
7:                  for v ∈ Ex_i do
8:                      if A'_{u,v} ∧ ¬A_{u,v} then
9:                          A' ← A' ∪ getEdges(i, u, v)        ▷ Добавление новых рёбер
10:         A → A'
11:     return A
```

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

Далее, внешний цикл повторяется, пока матрица смежности A меняется (то есть пока добавляются новые рёбра). На каждой итерации считается A' — транзитивное замыкание A. После этого находятся все новые пути вида $\langle$стартовое состояние$\rangle \rightsquigarrow \langle$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) отработает суммарно за $\mathcal{O}(|V|^2)$ (так как 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

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 — $(1, -1)$.

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

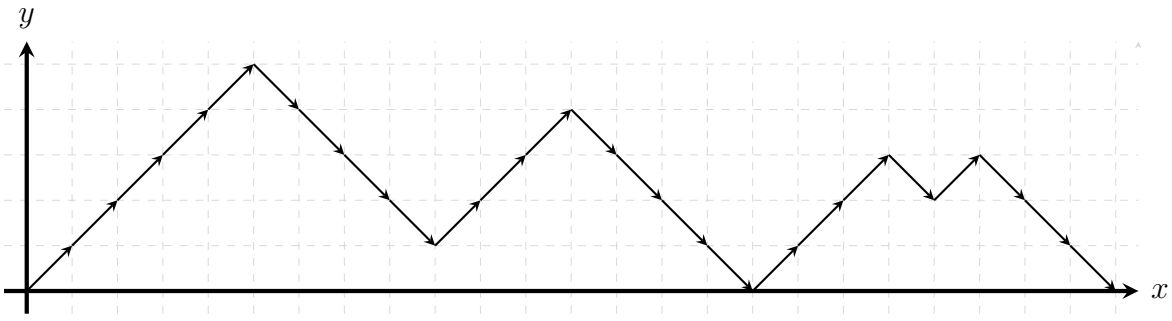


Рис. 14: Путь Дика для строки ((((()))(((())))(((()))

Для построения алгоритма воспользуемся следующим результатом:

Лемма 4.1 (Дюлеаж и Лоран [19])**.** *Для языка L определим рациональный индекс языка L — $p_L(n)$* как максимальную длину кратчайшего слова в $L \cap K$ по всем регулярным языкам K, задаваемым НКА с $\leqslant n$ состояниями.*

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

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 $\mathcal{O}(|V|^2)$.*

*Формально, $p_L(n) = \max\{\min\{|w|\colon w \in L \cap K\}, K \in Rat_n(X), L \cap K \neq \varnothing\}$,
где $Rat_n(X)$ — регулярные языки над алфавитом X, распознаваемые НКА с $\leqslant n$ состояниями.

†Точная оценка — $2n^2 + 4n$

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 (считающим баланс), 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.

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

Заметим, что для этого придётся видоизменить грамматику, для её стандартного вида ($\mathcal{D}_1 \colon S \to SS \mid (S) \mid \varepsilon$) 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 $S \to (S)$), так что потребуются все k итераций.

Другое представление языка $\mathcal{D}_1$ основано на двух 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 $K = \lceil \log(2n^2 + 4n) \rceil$ (то есть 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 $(^{2^i}$, D_i сжимает пути вида $)^{2^i}$:

$$
\begin{aligned}
U_0 &\to (\\
U_1 &\to U_0\, S\, U_0 \\
U_2 &\to U_1\, S\, U_1 \\
&\dots \\
U_K &\to U_{K-1}\, S\, U_{K-1}
\end{aligned}
$$

В продукции для U_i есть S между U_{i-1} и U_{i-1} — она нужна, так как U_i ищет не строго возрастающие пути, а разрешает им некоторое время “идти прямо” (по-

лучаются уже такие пути Моцкина [20], а не Дика, в которых рёбра с метками S соответствуют горизонтальным рёбрам).

Продукции для D_i устроены аналогично.

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

$$\mathcal{D}_1 \colon S \to U_0\, S\, D_0 \mid U_1\, S\, D_1 \mid \ \ldots\ \mid U_K\, S\, D_K \mid SS \mid \varepsilon \tag{1}$$

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

Так, продукция $S \to SS$ 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 $S \to S^*$.

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

4.2.2. Пример

На рисунках 15-21 приведён пример работы алгоритма 4 для языка $\mathcal{D}_1$ при поиске конкретного пути — строки ‘((((()))((()))(((())))’ (из примера 14).

(Для краткости используется немодифицированная грамматика 1, разбиение нетерминала S на K штук добавит по одной лишней итерации на продукцию $S \to SS$).

Для каждой итерации обозначены только нужные рекурсивные рёбра (то есть те, которые будут использованы на следующей или более поздних итерациях).

На первой проводятся все S-петли и все рёбра с метками U_0 и D_0.

На второй итерации проводятся рёбра с метками U_1 и D_1 (по правилам $U_1 \to U_0\, S\, U_0$ и $D_1 \to D_0\, S D_0$), а также два S-ребра по правилу $S \to U_0\, S\, D_0$.

На третьей итерации проводится S-ребро по правилу $S \to U_1\, S\, D_1$, и ещё одно S-ребро по правилу $S \to SS$.

На четвёртой итерации проводится два S-ребра по правилам $S \to U_2\, S\, D_2$ и $S \to U_1\, S\, D_1$.

На итерациях 5 и 6 проводится по одному S-ребру по правилам $S \to S\, S$ и $S \to U_0\, S\, D_0$ соответственно.

На последней итерации проводится S-ребро, соответствующее всему пути.

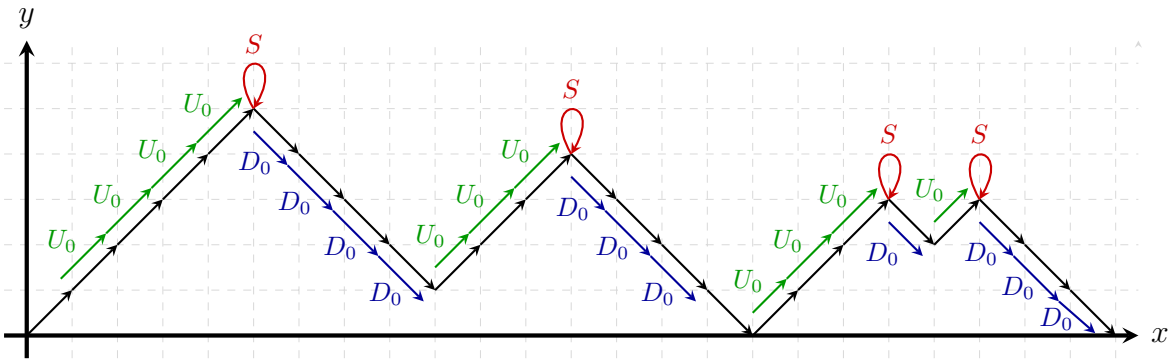


Рис. 15: Итерация 1

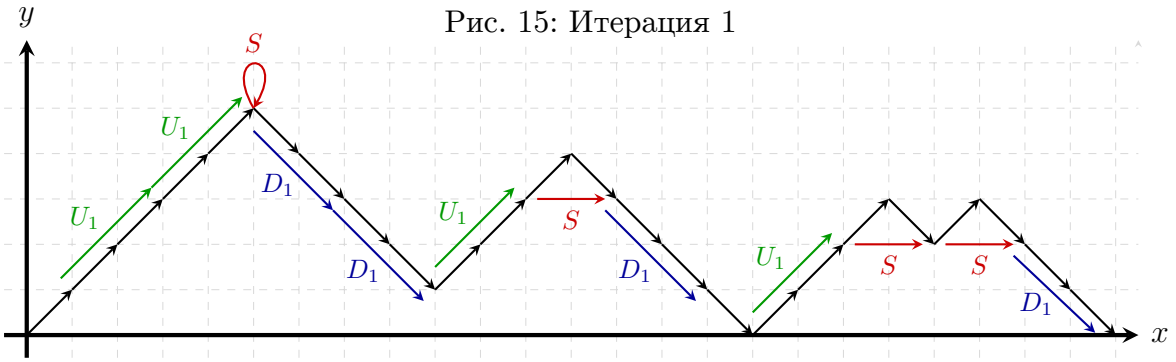


Рис. 16: Итерация 2

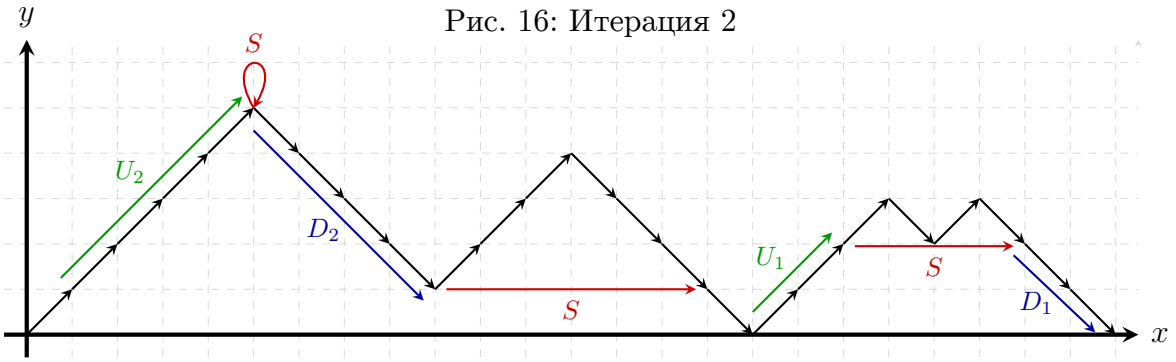


Рис. 17: Итерация 3

Рис. 18: Итерация 4

Рис. 19: Итерация 5

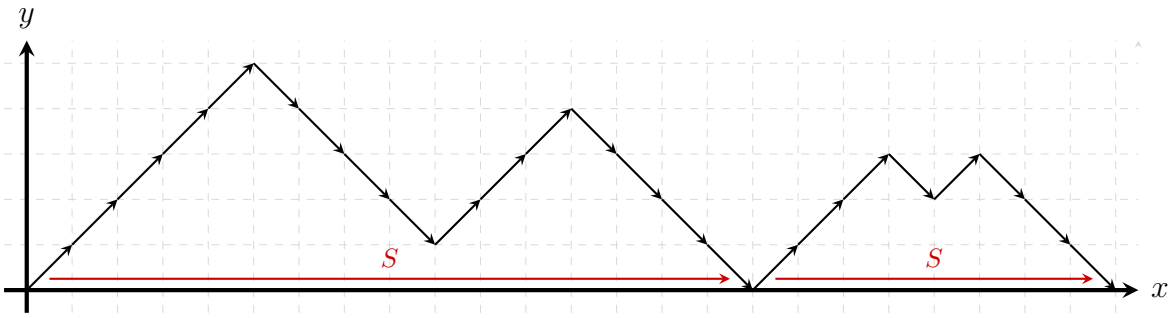


Рис. 20: Итерация 6

Рис. 21: Итерация 7

4.2.3. Корректность алгоритма

Посмотрим на нетерминал S (опр. 1). 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 строки с балансом 2^i и префиксными балансами $\geqslant 0$
2. D_i выводит строки с балансом (-2^i) и с префиксными балансами $\geqslant -2^i$
3. S выводит корректные правильные скобочные последовательности

База очевидна, переход тоже.

4.2.4. Время работы

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

Доказательство. Время работы алгоритма 4 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), так как на каждой итерации считалось транзитивное замыкание. Заметим, что на самом деле достаточно находить транзитивное замыкание отдельно в каждой компоненте РКА (так как разные компоненты вообще

не связны). Поскольку компоненты РКА грамматики константны, 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.

Покажем, что все горы исходной строки сожмутся за $2K$ итераций. Длина горы l раскладывается на сумму степеней двойки: $l = a_0 \cdot 1 + a_1 \cdot 2 + \cdots + a_K \cdot 2^K$, где $a_i \in \{0, 1\}$. Покажем по индукции, что после $(2i + 2)$-ого шага будет сжата в S-ребро верхушка горы длины $a_0 \cdot 1 + a_1 \cdot 2 + \cdots + a_{i-1} \cdot 2^{i-1}$. База очевидна, переход: знаем, что на $2i$-ом шаге сжалась верхушка размера $a_0 \cdot 1 + a_1 \cdot 2 + \cdots + a_{i-2} \cdot 2^{i-2}$, то есть остаётся только сжать в S-ребро строку вида $({}^{2^{i-1}}\, S\,)^{2^{i-1}}$. Заметим, что к $2i$-ому шагу строка ${}^{2^{i-1}}$ уже будет сжата в U_{i-1}-ребро, а $)^{2^{i-1}}$ — в ребро с меткой D_{i-1}. Остаётся только применить продукции $S_{i-1} \to U_{i-1}\, S\, D_{i-1}$ и $S \to S_{i-1}$.

После того, как сжались горы исходной строки, начнут сжиматься горы, которые не могли сделать этого раньше. Почему не могли? Потому что между подъёмом и спуском горы было не было S-ребра, а сейчас появилось. Любой одиночный пик сжался бы за одну фазу (= $2K$ итераций), значит между подъёмом и спуском новой горы было как минимум два пика (горы) предыдущей фазы, сжатые в одно S-ребро продукцией $S \to (\bigcup S_i)^*$.

То есть после каждой такой фазы сжимания число пиков/гор уменьшается хотя бы в 2 раза. Так как 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 сжимание вертикальных U_i и D_i может также накладываться на предыдущую фазу сжатия гор.

□

Замечание. Данный алгоритм будет работать также и для языка полу-Дика (semi-Dyck language), задаваемого грамматикой $S \to SS \mid (S) \mid)S(\mid \varepsilon$, то есть языка строк с нулевым балансом.

Понятно, что алгоритм придётся видоизменить, а именно, добавить в грамматику продукции вида $S \to D_i\, S\, U_i$ (чтобы сжимать перевёрнутые горы).

Для доказательства оценки на время работы нужно иметь про язык полу-Дика факт, аналогичный лемме 4.1. В своей работе [9] Боассон и др. доказывают оценку в $\mathcal{O}(n^3)$ на рациональный индекс $p_{\mathcal{D}_1}(n)$ языка Дика. Ровно такие же рассуждения срабатывают и для языка полу-Дика, так что алгоритм 4 работает за $\mathcal{O}(|V|^\omega \log^3 |V|)$ и для него.

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

В данной главе была предложена модификация алгоритма 1, основанная на идее, что можно не поддерживать инкрементальное транзитивное замыкание, а каждый раз пересчитывать обычное. Далее этот алгоритм был применён для языка Дика на одном типе скобок. Для этого для языка $\mathcal{D}_1$ была построена специальная грамматика, дающая малое число итераций пересчёта транзитивного замыкания.

5. Смешанный языка Дика

5.1. Алгоритм для смешанного языка Дика

Смешанный язык Дика $\mathcal{D}_i \odot \mathcal{D}_j$ является пересечением двух контекстно-свободных языков, а именно, $\mathcal{D}_i \odot \mathcal{D}_j = L_1 \cap L_2$, где $L_1 \colon S \to SS \mid ({}_1S)_1 \mid \ldots \mid ({}_iS)_i \mid \varepsilon \mid [{}_1 \mid]_1 \mid \ldots \mid [{}_j \mid]_j$ и $L_2 \colon T \to TT \mid [{}_1T]_1 \mid \ldots \mid [{}_jT]_j \mid \varepsilon \mid ({}_1 \mid)_1 \mid \ldots \mid ({}_i \mid)_i$, то есть двух языков Дика, которые игнорируют скобки второго типа.

Однако контекстно-свободные языки не замкнуты относительно пересечения: при нём получается конъюнктивный язык [44], задача достижимости для которого является алгоритмически неразрешимой [25]. Более того, она неразрешима уже для языка $\mathcal{D}_2 \odot \mathcal{D}_2$ даже на двунаправленных графах [36].

Однако задача достижимости для языка $\mathcal{D}_1 \odot \mathcal{D}_1$ на двунаправленных графах вполне разрешима, причём за полиномиально время.

В этой главе будет приведён улучшение алгоритма Ли и др. [36] решения этой задачи, основанное на идее пересечения языков.

Замечание. Здесь и далее будем считать, что первый язык Дика — язык круглых ПСП ($\mathcal{D}_p \colon S \to SS \mid (S) \mid \varepsilon$), второй — язык квадратных ПСП ($\mathcal{D}_b \colon T \to T \mid [T] \mid \varepsilon$).

Заметим, что слово принадлежит $\mathcal{D}_p \odot \mathcal{D}_b$, тогда и только тогда, когда на любом его префиксе баланс обоих типов скобок неотрицателен, а конечный баланс равен нулю. Баланс строки будем записывать как (p, b), где p — баланс круглых скобок, а b — квадратных.

Для решения понадобится следующий вспомогательный факт:

Лемма 5.1. *Ли и др. [36] В двунаправленном графе, если между парой вершин (u, v) существует какой-то $\mathcal{D}_1 \odot \mathcal{D}_1$ путь, то существует и такой путь, на котором в любой момент времени вложенность хотя бы одного типа скобок не превышает $6|V|$.*

Нарисуем (рис. 22) такой (как в утверждении леммы 5.1) путь на плоскости, где первая координата — баланс круглых скобок, вторая — баланс квадратных (то есть просто p и b). Тогда во-первых, весь путь будет проходить в первой четверти. Во-вторых, он не будет заходить (найдётся такой, что не будет заходить, и мы ищем именно его) в сектор $[6|V| + 1, +\infty) \times [6|V| + 1, +\infty)$

То есть путь можно искать следующего вида: он в основном проходит внутри квадрата $[0, 6|V|] \times [0, 6|V|]$, иногда выходя из него, но только по одной координате (то есть либо в сектор $[0, 6|V|] \times [6|V| + 1, +\infty)$, либо в сектор $[6|V| + 1, +\infty) \times [0, 6|V|]$) (обозначены на рис. 22 красным цветом).

Тогда можно искать отдельно две эти сущности: куски путей внутри квадрата $[0, 6|V|] \times [0, 6|V|]$, и куски путей снаружи.

Вооружившись этим знанием, построим алгоритм:

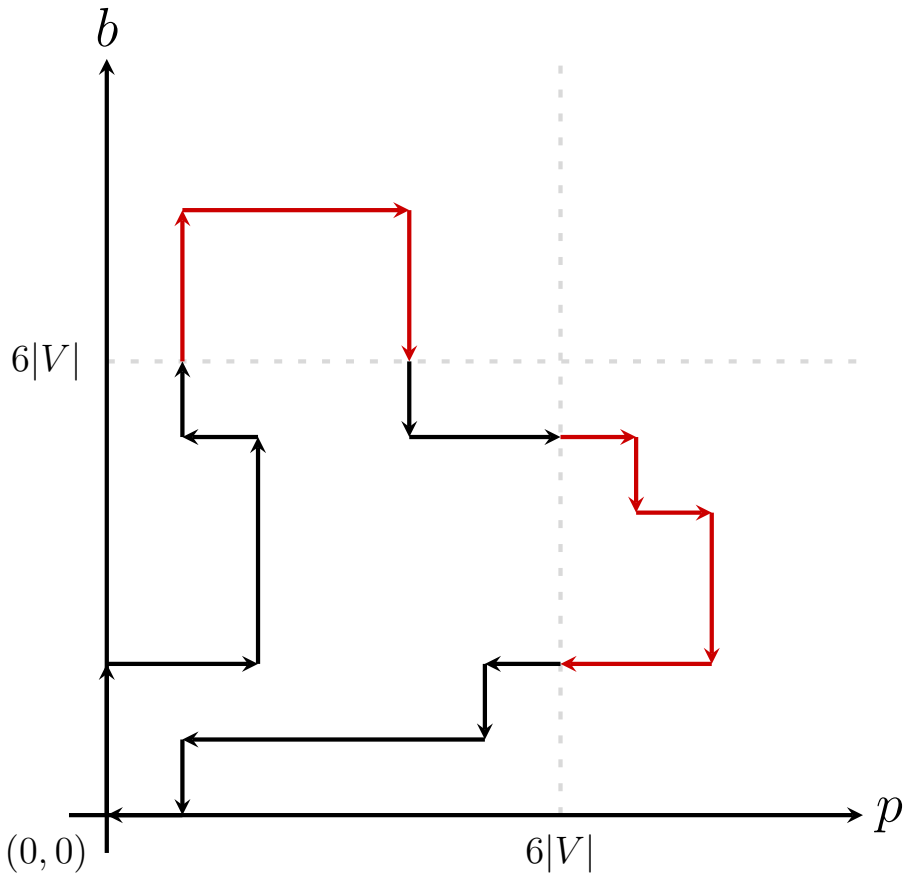


Рис. 22: Путь в координатах баланса

1. **Часть пути, лежащая в квадрате $[\mathbf{0}, \mathbf{6|V|}] \times [\mathbf{0}, \mathbf{6|V|}]$**

   Внутри квадрата ограничена вложенность обоих типов скобок. Помним из прошлой главы (замечание 4.2), что в случае ограниченной вложенности язык получается регулярным. Строим автоматы для таких регулярных языков: $\mathcal{D}_p^{6|V|}$ и $\mathcal{D}_b^{6|V|}$, оба размера $\mathcal{O}(n)$.

   Пересекая оба этих языка и входной граф получаем автомат, состояние в котором — тройка $\langle v, p, b\rangle$ (вершина и два баланса). Размер автомата: $\mathcal{O}(|V|^3)$ вершин, $\mathcal{O}(|E||V|^2)$ рёбер.

   Также в этом автомате нужно провести ε-рёбра, соответствующие путям, выходящим за границы квадрата. Такие пути идут из состояния $(u, 6|V|, b_1)$ в состояние $(v, 6|V|, b_2)$ (и аналогично для другого типа скобок). Таких рёбер может быть столько, сколько есть различных пар $(u, b_1)/v, b_2$, то есть $\mathcal{O}(|V|^4)$.

   После этого нужно решить задачу достижимости для полученного графа-автомата. Покажем, что он получается неориентированным: из-за двунаправленности графа ребро $(u, b, p) \to (v, b+1, p)$ существует тогда и только тогда, когда и обратное ему. То же и с ε-рёбрами: если есть путь, с балансом $(0, j-i)$, то есть и путь в обратную сторону с балансом $(0, i-j)$.

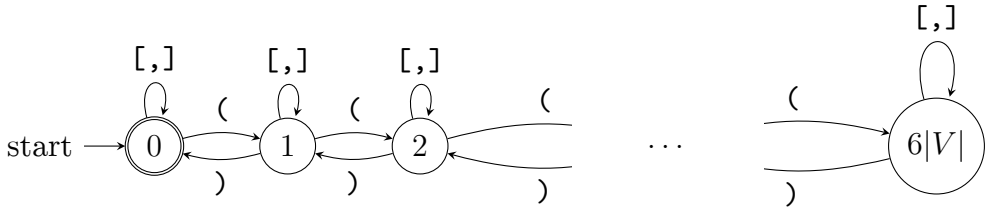


Рис. 23: Автомат для языка $\mathcal{D}_p^{6|V|}$

В неориентированном графе выделяем компоненты связности, теперь из вершины u есть $\mathcal{D}_p \odot \mathcal{D}_p$-путь, если $(u, 0, 0)$ и $(v, 0, 0)$ лежат в одной компоненте.

Итого, эта часть алгоритма работает за dfs по графу-произведению, то есть за $\mathcal{O}(|V|^4)$.

2. **Часть пути, лежащая в $[\mathbf{0}, \mathbf{6|V|}] \times [\mathbf{6|V|+1}, +\infty)$**

   Хотим найти все пути вида $(u, 6|V|, b_1) \rightsquigarrow (v, 6|V|, b_2)$, то есть пути, образующие правильные круглые ПСП, а по квадратным дающие разницу балансов $b_2 - b_1$. При этом квадратный баланс не должен подниматься выше $6|V|$ и опускаться ниже 0 (то есть важно его стартовое значение).

   Будем решать отдельно для каждого стартового квадратного баланса b_1. Снова делаем это через пересечения языков. Первый — входной граф, второй — круглые ПСП (язык $\mathcal{D}_p$ из 5.1). Третий — регулярный язык, считающий квадратный баланс $\mathcal{D}_b^{b_1, 6|V|}$, его единственное отличие от языков в прошлом пункте в том, что баланс может опускаться не до 0, а до $-b_1$ (и подниматься до $6|V| - b_1$, а не до $6|V|$).

   Пересекаем языки в хитром порядке: сначала первый и третий (то есть входной граф и регулярный язык), получим опять-таки граф пар (v, p) — вершина и круглый баланс — на $\mathcal{O}(|V|^2)$ вершинах и $\mathcal{O}(|V||E|)$ рёбрах. Дальше нужно пересечь его со вторым языком, который просто является языком Дика, то есть решить для него задачу Диковой достижимости.

   Заметим, что граф $L(G) \cap \mathcal{D}_b^{b_1, 6|V|}$ получится также двунаправленным:

   - $u \xrightarrow{[} v$ в $G \Rightarrow$ рёбра $\langle u, b\rangle \xrightarrow{\varepsilon} \langle v, b+1\rangle$ в пересечении
   - $u \xrightarrow{]} v$ в $G \Rightarrow$ рёбра $\langle u, b\rangle \xrightarrow{\varepsilon} \langle v, b-1\rangle$ в пересечении
   - $u \xrightarrow{[} v \Leftrightarrow v \xrightarrow{]} u$, так что ε-рёбра двунаправлены
   - $u \xrightarrow{(} v$ в $G \Rightarrow$ рёбра $\langle u, b\rangle \xrightarrow{(} \langle v, b\rangle$ в пересечении
   - $u \xrightarrow{)} v$ в $G \Rightarrow$ рёбра $\langle u, b\rangle \xrightarrow{)} \langle v, b\rangle$ в пересечении
   - $u \xrightarrow{(} v \Leftrightarrow v \xrightarrow{)} u$, так что (и)-рёбра также двунаправлены

   Задача Диковой достижимости на двунаправленных графах решается за время $\mathcal{O}(|V|\alpha(|V|) + |E|)$ [12] (в нашем случае, получится $\mathcal{O}(|V||E|)$. Решаем её, теперь

путь $(u, 6|V|, b_1) \rightsquigarrow (v, 6|V|, b_2)$ существует тогда и только тогда, когда $(v, b_2 - b_1)$ Диково достижимо из $(u, 0)$ в полученном графе.

Получаем алгоритм за $\mathcal{O}(|V||E|)$ для каждого b_1, итого $\mathcal{O}(|V|^2|E|)$.

Теорема 5.1. *Для задачи $\mathcal{D}_1 \odot \mathcal{D}_1$-достижимости существует алгоритм с временем работы $\mathcal{O}(|V|^4)$.*

5.2. Выводы и результаты по главе

В данной главе был приведена модификация алгоритма Ли и др. [36] решения задачи достижимости для смешанного языка Дика $\mathcal{D}_1 \odot \mathcal{D}_1$ на двунаправленных графах, в которой применялась идея пересечения языков: входной граф пересекался с двумя языками Дика, которые в обоих случая были ограничены так, что хотя бы один из них становился регулярным.

Заключение

Главным результатом данной работы является новый подход для разработки частных решений для задачи поиска путей с контекстно-свободными ограничениями. Подход заключается в сведении задачи поиска путей с контекстно-свободными ограничениями к решению задачи достижимости на рекурсивном автомате, распознающем пересечение языков входной грамматики и входного графа. В общем случае задача достижимости для РКА решается поддержанием инкрементального транзитивного замыкания и имеет ту же асимптотику, что и все остальные решения задачи CFPQ (а именно, $\mathcal{O}(|V|^3)$), однако оно может быть модифицировано для получения более быстрых решений в частных случаях.

Первой такой модификацией стало решение 2, основанное на поддержании неориентированного инкрементального транзитивного замыкания (вместо ориентированного). Такое решение будет, в частности, работать, если РКА пересечения является неориентированным, но не только. Более практичным применением такого алгоритма является решение задачи Диковой достижимости на двунаправленных графах (теорема 3.1).

Ещё одной модификацией является алгоритм 4, основанный на пересчёте транзитивного замыкания каждый раз с нуля, а не итеративном обновлением при добавлении очередного ребра. Такой алгоритм будет иметь субкубическое время работы, если число итераций пересчёта транзитивного замыкания является небольшим. Например, такое алгоритм имеет время работы $\mathcal{O}(|V|^\omega \log^3 |V|)$ для языка Дика на одном типе скобок (теорема 4.1).

Идея пересечения языком может быть применена не только для задачи поиска путей с контекстно-свободными ограничениями. Так, в главе 5 этот подход применяется для решения задачи $\mathcal{D}_1 \odot \mathcal{D}_1$-достижимости, то есть для языка, который не является контекстно-свободным.

В качестве продолжения этой работы возможно рассмотрение применения данного подхода для других частных случаев задачи поиска путей с контекстно-свободными ограничениями, например, для планарных графов.

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] Bastani Osbert, Anand Saswat, Aiken Alex. Specification inference using context-free language reachability // Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. — 2015. — P. 553–566.

[9] Boasson Luc, Courcelle Bruno, Nivat Maurice. The Rational Index: A Complexity Measure for Languages // SIAM Journal on Computing. — 1981. — Vol. 10, no. 2. — P. 284–296. — https://doi.org/10.1137/0210020.

[10] 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.

[11] 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.

[12] 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.

[13] Chaudhary Anoop, Faisal Abdul. Role of graph databases in social networks. — 2016. — 06.

[14] CFL-reachability in subcubic time : Rep. / Technical report, IBM Research Report RC24126 ; Executor: Swarat Chaudhuri : 2006.

[15] 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.

[16] Chistikov Dmitry, Majumdar Rupak, Schepper Philipp. Subcubic Certificates for CFL Reachability // arXiv preprint arXiv:2102.13095. — 2021.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] Hellings Jelle. Conjunctive Context-Free Path Queries. // ICDT. — 2014. — P. 119–130.

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

[27] Hellings Jelle. Querying for Paths in Graphs using Context-Free Path Queries. — 2016. — 1502.02242.

[28] Hopcroft John E., Ullman Jeffrey D. Set merging algorithms // SIAM Journal on Computing. — 1973. — Vol. 2, no. 4. — P. 294–303.

[29] Hopcroft John E, Ullman Jeffrey D. An introduction to automata theory, languages, and computation. — Upper Saddle River, NJ : Pearson, 1979.

[30] 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.

[31] Impagliazzo Russell, Paturi Ramamohan. On the complexity of k-SAT // Journal of Computer and System Sciences. — 2001. — Vol. 62, no. 2. — P. 367–375.

[32] 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.

[33] Karczmarz Adam. Data structures and dynamic algorithms for planar graphs : Ph. D. thesis / Adam Karczmarz ; PhD thesis, University of Warsaw. — 2018. — P. 9.

[34] 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.

[35] Kutrib Martin, Malcher Andreas, Schneider Christian. Finite Automata with Undirected State Graphs // Descriptional Complexity of Formal Systems / Ed.

by Stavros Konstantinidis, Giovanni Pighizzini. — Cham : Springer International Publishing, 2018. — P. 212–223.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] Milanova Ana, Huang Wei, Dong Yao. CFL-reachability and context-sensitive integrity types // Proceedings of the 2014 International Conference on Principles and Practices of Programming on the Java platform: Virtual machines, Languages, and Tools. — 2014. — P. 99–109.

[42] Modular shape analysis for dynamically encapsulated programs / Noam Rinetzky, Arnd Poetzsch-Heffter, Ganesan Ramalingam et al. // European Symposium on Programming / Springer. — 2007. — P. 220–236.

[43] 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.

[44] Okhotin Alexander. Conjunctive grammars // Journal of Automata, Languages and Combinatorics. — 2001. — Vol. 6, no. 4. — P. 519–535.

[45] On economical construction of the transitive closure of an oriented graph / Vladimir L'vovich Arlazarov, Yefim A Dinitz, MA Kronrod, IgorAleksandrovich Faradzhev // Doklady Akademii Nauk / Russian Academy of Sciences. — 1970.

[46] 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.

[47] Pratikakis Polyvios, Foster Jeffrey S, Hicks Michael. Existential label flow inference via CFL reachability // International Static Analysis Symposium / Springer. — 2006. — P. 88–106.

[48] RDF - Semantic Web Standards. — Accessed: 2020-05-15. Access mode: `https://www.w3.org/RDF/`.

[49] 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`.

[50] 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`.

[51] 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.

[52] SPARQL 1.1 overview. — Accessed: 2020-05-15. Access mode: `https://www.w3.org/TR/sparql11-query/`.

[53] Sagiv Mooly, Reps Thomas, Horwitz Susan. Precise interprocedural dataflow analysis with applications to constant propagation // Theoretical Computer Science. — 1996. — Vol. 167, no. 1-2. — P. 131–170.

[54] 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.

[55] 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.

[56] Scott Elizabeth, Johnstone Adrian, Hussain Shamsa Sadaf. Tomita-style generalised LR parsers // Royal Holloway University of London. — 2000.

[57] 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.

[58] Speeding up Slicing / Thomas Reps, Susan Horwitz, Mooly Sagiv, Genevieve Rosay // Proceedings of the 2nd ACM SIGSOFT Symposium on Foundations of Software Engineering. — SIGSOFT '94. — New York, NY, USA : Association for Computing Machinery, 1994. — P. 11–20. — Access mode: https://doi.org/10.1145/193173.195287.

[59] Sridharan Manu, Bodík Rastislav. Refinement-based context-sensitive points-to analysis for Java // ACM SIGPLAN Notices. — 2006. — Vol. 41, no. 6. — P. 387–400.

[60] 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.

[61] 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.

[62] 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.

[63] 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.

[64] Williams Virginia Vassilevska. On some fine-grained questions in algorithms and complexity // Proceedings of the ICM / World Scientific. — Vol. 3. — 2018. — P. 3431–3472.

[65] Xu Guoqing, Rountev Atanas, Sridharan Manu. Scaling CFL-reachability-based points-to analysis using context-sensitive must-not-alias analysis // European Conference on Object-Oriented Programming / Springer. — 2009. — P. 98–122.

[66] 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.

[67] 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.

[68] Yannakakis Mihalis. Hierarchical state machines // IFIP International Conference on Theoretical Computer Science / Springer. — 2000. — P. 315–330.

[69] 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.

[70] 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.

[71] 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.

[72] Yuval Gideon et al. AN ALGORITHM FOR FINDING ALL SHORTEST PATHS USING $N^{2.81}$ INFINITE-PRECISION MULTIPLICATIONS. — 1976.

[73] 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.

[74] Zhang Qirun. Conditional Lower Bound for Inclusion-Based Points-to Analysis // arXiv preprint arXiv:2007.05569. — 2020.

[75] 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.