Синтаксический анализ ациклических графов с использованием булевых грамматик

[1] *С.В. Григорьев <s.v.grigoriev@spbu.ru>*

[2] *Е.Н. Шеметова <katyacyfra@gmail.com>*

[1] *Санкт-Петербургский государственный университет,*

199034, Россия, г. Санкт-Петербург, Университетская наб., д. 7/9.

[2] *Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики (Университет ИТМО),*

197101, Россия, г. Санкт-Петербург, Кронверкский пр., 49

Аннотация.

Ключевые слова: синтаксический анализ графов; булевы грамматики; матричные операции; ациклический граф; топологическая сортировка; произведение матриц

DOI:

Для цитирования:

1. Введение

Графовая модель данных широко используется для решения задач, данные в которых тесно связаны между собой отношениями. Эти отношения как правило составляют сложную иерархию. Примеры сфер использования графовых моделей данных – графовые базы данных, биоинформатика, моделирование и анализ социальных сетей [1], статический анализ кода и многое другое.

Так как роль графов — представить отношения между различными объектами, часто бывает необходимо вычислять запросы к большим графам с целью выявления сложных зависимостей между их вершинами. Естественным способом описать отношения между графовыми объектами является использование формальной грамматики над метками рёбер графа. Популярными являются запросы, использующие регулярные грамматики. Также существуют запросы с использованием контекстно-свободных грамматик, позволяющие искать более сложные отношения по сравнению с запросами, использующими регулярные грамматики. Стоит отметить булевы грамматики, расширяющие класс контекстно-свободных грамматик такими теоретико-множественными операциями, как конъюнкция и отрицание [3]. С помощью булевых грамматик можно выразить более широкий класс языков, а значит, формулировать более сложные запросы к графам. Помимо этого, булевы грамматики естественным образом позволяют применять технику гибкого синтаксического анализа (agile parsing) [4], которая может быть полезна для решения более сложных задач, например, в статическом анализе кода [5].

В данной работе предложен алгоритм, вычисляющий аппроксимацию решения задачи синтаксического анализа ациклических графов с использованием реляционной семантики запросов и булевых грамматик.

Работа организована следующим образом. В разделе 2 даны основные определения, связанные с задачей синтаксического анализа графов, и рассмотрены основные подходы данной области; в разделе 3 проанализированы существующие решения данной задачи; в разделе 4 представлена адаптация алгоритма Охотина [6], находящая аппроксимацию решения задачи синтаксического анализа ациклических графов с использованием булевых грамматик; также доказана корректность применения данного алгоритма для поставленной задачи; в разделе 5 работа предложенного алгоритма разобрана на примере; заключение и направления будущих исследований приведены в разделе 6.

2. Основные определения и подходы к решению

Для начала определим задачу синтаксического анализа графа с использованием реляционной семантики запросов.

Рассмотрим ориентированный ациклический граф $D = (V, E)$ и формальную грамматику G. Пусть у каждого ребра графа есть метка, множество всех меток обозначим Σ. Тогда каждый путь в D будет обозначать слово над алфавитом из Σ, полученное конкатенацией меток рёбер, включенных в этот путь.

Для графа D и формальной грамматики $G = \{\Sigma, N, P, S\}$ мы обозначим отношения $R_A \subseteq V \times V$ для каждого $A \in N$ следующим образом:

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

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

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

В качестве формальной грамматики G будут использованы булевы грамматики [3]. Булева грамматика является классом формальных грамматик, расширяющим класс контекстно-свободных грамматик с помощью булевых операций конъюнкции и отрицания.

Булева грамматика формально определена следующим образом:

$G = \{\Sigma, N, R, S\}$, где G – булева грамматика, Σ – терминальный алфавит, N – нетерминальный алфавит (множество нетерминальных символов $\{A_1, A_2, \dots, A_n\}$), S – стартовый нетерминал, R – множество правил грамматики.

Тогда правила булевой грамматики можно представить в виде:

$$A \to \alpha_1 \& \alpha_2 \& \dots \& \alpha_m \& \neg\beta_1 \& \neg\beta_2 \& \dots \& \neg\beta_n\ ,$$

где A – нетерминал, $m + n \geq 1$, $\alpha_1, \alpha_2, \dots \alpha_m, \beta_1, \beta_2, \dots \beta_n \in (\Sigma \cup N)^*$.

Если в правилах грамматики отсутствуют отрицания, то грамматика называется конъюнктивной (расширение контекстно-свободной грамматики операцией конъюнкции) [].

Булева грамматика $G = \{\Sigma, N, R, S\}$ находится в *двоичной нормальной форме*, если каждое правило в R находится в одной из следующих форм:

- $A \to B_1 C_1 \& \dots \& B_m C_m \& \neg D_1 E_1 \& \dots \& \neg D_n E_n \& \neg\varepsilon\ (m \geq 1, n \geq 0)$

- $A \to a \quad (a \in \Sigma)$
- $S \to \varepsilon$ (только если S не появляется в правых частях правил)

3. Существующие решения

Решение задачи с использованием реляционной семантики и регулярных грамматик представлены в работах [], []. Также существует ряд алгоритмов для синтаксического анализа графов с использованием реляционной семантики запросов и КС-грамматик []. Также в работах [7, 8] представлены алгоритмы синтаксического анализа графов, использующий реляционную семантику запросов и конъюнктивные грамматики.

Представленные алгоритмы работают с графами произвольной структуры. Если же исходный граф является ациклическим, то можно воспользоваться известными свойствами ациклических графов, чтобы предоставить более производительный алгоритм для решения задачи синтаксического анализа для данной структуры графа. Пример подобного подхода рассмотрен в работе [9].

Вышеперечисленные алгоритмы работают только с контекстно-свободными и конъюнктивными грамматиками. Таким образом, задача построения алгоритма синтаксического анализа графов, использующего реляционную семантику запросов и работающего с произвольными булевыми грамматиками, является открытой.

4. Алгоритм

В работе [6] предложен быстрый алгоритм синтаксического анализа, основанный на матричных операциях и обобщённый для булевых грамматик. В данном разделе будет предложено расширение этого алгоритма для аппроксимации решения задачи синтаксического анализа ациклических графов и показана его корректность.

4.1 Аппроксимация решения

Рассматриваемый ниже алгоритм основан на матричных операциях. Так как необходимо найти тройки (A, i, j) для всех $A \in N$, результатом работы алгоритма на графе с n вершинами будет матрица размером $n \times n$, в каждой из ячеек которой будут содержаться подмножества нетерминалов. На каждой итерации алгоритма, подмножества нетерминалов могут объединяться с помощью операции конъюнкции, если во входной грамматике существует соответствующее правило. Поэтому важно объеди-нять только те нетерминалы, которые соответствуют одному и тому же уникальному пути между парой вершин графа. В случае, если входной ациклический граф является деревом, между любой парой вершин графа будет существовать всего один путь. Если

же граф не является деревом, то между одной парой вершин может бытьв худшем случае 2^{n-2} путей, поэтому хранить и обрабатывать данные для каждого пути отдельно неэффективно. Так как информацию о всех путях из вершины i в вершину j эффективно хранить в одной ячейке, то с помощью конъюнкции могут быть объединены нетерминалы, соответствующие разным путям из i в j, но при этом будут объединены также нетерминалы, соответствующие одному и тому же пути, что и требуется для решения задачи. То же самое работает и для отрицаний в правиле грамматики: получив множества пар нетерминалов для одной пары вершин, необходимо проверить отсутствие в этом множестве определенных пар нетерминалов (согласно правилам грамматики). Так как неизвестно, какому конкретно пути соответствуютнайденные нетерминалы, ответ будет неточным. Например, пусть в грамматике есть правило $A \to AB\&BC\&\neg DC$, а найденное множество пар нетерминалов для парывершин (i,j) - $\{AB, BC, DC\}$. Тогда, после применения упомянутого правила грамматики, ответ будет содержать утверждение $(i,j) \in R_A$. Но, полученное множество $\{AB, BC, DC\}$ может выводить один и тот же уникальный путь между вершинами i и j, тогда из-за отрицания DC, не будет существовать пути из вершины i в вершину j, образующего строку, выводимую из нетерминала A. Алгоритм рассматривает всевозможные сочетания пар нетерминалов, в результате могут появится лишние вершины (i,j), если в графе не существует пути из i в j, соответствующего рассмотренным подмножествам пар и правилу для них.
Поэтому приведенный ниже алгоритм будет вычислять аппроксимацию сверху для отношений R_A (надмножество необходимого множества).

4.2 Описание алгоритма

Пусть n - число вершин в помеченном графе D, $G = \{\Sigma, N, R, S\}$ - булева грамматика в двоичной нормальной форме. Алгоритм использует следующие структуры данных:

1) Таблица парсинга T размером $n \times n$ представляющая собой верхнетреугольную матрицу, где каждый элемент $T_{i,j}$ является множеством нетерминалов.

Пусть $X \in (2^N)^{m \times l}$ и $Y \in (2^N)^{l \times n}$ – матрицы подмножеств множества нетерминалов N, а m, l, n – натуральные числа, большие единицы. Тогда *произведением матриц* $X \times Y$ будет матрица $Z \in (2^N)^{m \times n}$, такая, что каждый её элемент $Z_{i,j}$ вычисляется по следующей форме:

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

2) Таблица P, каждый элемент $P_{i,j}$ которой принадлежит множеству пар нетерминалов $N \times N$.

Используя значения $P_{i,j}$, можно получить набор нетерминалов, генерирующих строку:

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

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

$$f(P) = \bigcup_{k=1}^{2^{|P|}} \{A \mid \exists\, A \to B_1 C_1 \& \ldots \& B_m C_m \& \neg D_1 E_1 \& \ldots \& \neg D_{m'} E_{m'} \in R : (B_t, C_t) \in P^k \text{ и } (D_t, E_t) \notin P^k \text{ для } \forall t,$$

где P^k - подмножество множества P.

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

В алгоритме Охотина элементы матрицы P рассчитываются группами с помощью выше определенного произведения подматриц из таблицы T, дающих аналогичный результат с поэлементным умножением. Схема расположения подматриц представлена на рисунках 1 и 2.

Алгоритм состоит из двух рекурсивных процедур:

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

2) $complete(l, m, l', m')$ – определена для $l \le m < l' \le m'$ при $m - l = m' - l'$ и $m - l$ без потери общности являющимися степенью двойки.

Четыре входных параметра процедуры обозначают координаты подматрицы матрицы T, содержащей все элементы $T_{i,j}$, где $l \le i < j < m$ и $l' \le j < m'$. Они обозначают пути в графе, номер начала которых лежит между l и m, а конец между l' и m'. Так как ранее уже посчитаны $T_{i,j}$ для $l \le i < j < m$ и $l' \le i < j < m'$, а также $P_{i,j}$ для $l \le i < m$ и $l' \le j < m'$, процедура $complete(l, m, l', m')$ получает значения $T_{i,j}$ для $l \le i < m$ и $l' \le j < m'$.

Так как основная структура данных, с которой работает алгоритм, является верхнетреугольными матрицами, то тогда должен быть задан линейный порядок на вершинах графа D, такой, что любое ребро ведет от вершины с меньшим номером к вершине с большим номером. Если D является ациклическим графом, то данного свойства легко добиться с помощью топологической сортировки графа. Алгоритм Тарьяна [10] позволяет

осуществлять топологическую сортировку за линейное от количества вершин графа время.

Полный псевдокод алгоритма приведен в листинге 1.

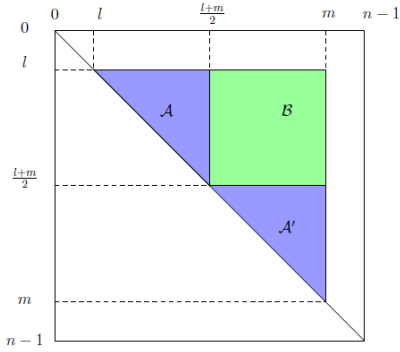

Рис. 1. Расположение подматриц для расчёта процедурой compute элементов таблицы P.

Fig. 1. Submatrices for calculating elements of P by «compute» procedure

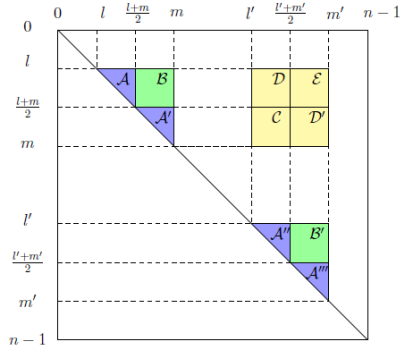

Рис. 2. Расположение подматриц для расчёта процедурой complete элементов таблицы P.

Fig. 2. Submatrices for calculating elements of P by «complete» procedure

```
/*  D – помеченный граф
/*  G – входная булева грамматика
1:  Main:
2:  n ⟵ число вершин в D
3:  E ⟵ {(i, j, a) | ребро из вершины i в j, помеченное символом a входит во
    множество рёбер D}
4:  N ⟵ множество нетерминалов грамматики G
5:  R ⟵ множество правил грамматики G
6:  T ⟵ пустая матрица размером n × n
7:  P ⟵ пустая матрица размером n × n
8:  D ⟵ TopologicalSorting(D)
9:  for each (i, j, a) ϵ E
10:   T_{i,j} ⟵ {A | A → a ϵ R}
11: compute(0, n - 1)

12: compute(l, m):
13: if m – l ≥ 4 then
14:   compute(l, (l+m)/2)
15:   compute((l+m)/2, m)
16: complete(l, (l+m)/2, (l+m)/2, m)

17: complete(l, m, l', m')
18: if (l, m, a) ϵ E  and m < l' then
19:   T_{l,l'} ⟵ f(P_{l,l'})
20: else if m-l > 1 then
21:   B ⟵ (l, (l+m)/2, (l'+m')/2, m')
22:   B' ⟵ ((l+m)/2, m, l', (l'+m')/2)
23:   C ⟵ ((l+m)/2, m, l', (l'+m')/2)
24:   D ⟵ (l, (l+m)/2, l', (l'+m')/2)
25:   D' ⟵ ((l+m)/2, m, (l'+m')/2, m')
26:   E ⟵ (l, (l+m)/2, (l'+m')/2, m')
27:   complete(C)
28:   P_D ⟵ P_D ∪ (T_B × T_C)
29:   complete(D)
30:   P_{D'} ⟵ P_{D'} ∪ (T_C × T_{B'})
31:   complete(D')
32:   P_E ⟵ P_E ∪ (T_B × T_{D'})
33:   P_E ⟵ P_E ∪ (T_D × T_{B'})
34:   complete(E)
```

Листинг. 1. Алгоритм синтаксического анализа ациклических графов.

Listing. 1. DAG-parsing algorithm

4.3 Время работы и корректность алгоритма

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

Теорема 1. (*Корректность алгоритма для синтаксического анализа ациклических графов*). Пусть даны помеченный ациклический граф $D = (V, E)$ и булева грамматика $G = (\Sigma, N, R)$. Тогда для любых вершин i, j и для любого нетерминала $A \in N$, если $(i, j) \in R_A$, то $A \in T_{i,j}$.

Доказательство.

Рассмотрим такие вершины i, j, что $(i, j) \in R_A$ и существует $i\pi j$, такой, что $l(\pi) \in L(G_A)$ (т.е. существует дерево разбора для строки $l(\pi)$ и грамматики G с корнем в нетерминале A). Докажем индукцией по высоте дерева разбора строки $l(\pi)$, что $A \in T_{i,j}$.

База индукции. Если (i, j) - ребро графа (высота дерева разбора равна 1), то алгоритм корректен, исходя из инициализации матрицы T (строки 9-10 листинга 1).

Индукционный переход. Предположим, утверждение верно для всех деревьев разбора высотой h. Докажем, что теорема верна для деревьев разбора высотой $h + 1$. Рассмотрим дерево разбора для $l(\pi)$ высотой $h + 1$. Так как грамматика в нормальной форме, то у данного дерева будут поддеревья, выводящие подстроки $l(\pi)$ (возможно пересекающиеся). По свойству топологической сортировки для всех индексов k, m этих подстрок (начала и конца соответствующих путей) выполняется неравенство $i \leq k < m \leq j$. $T_{i,j}$ вычисляется как $f(P_{i,j})$. $P_{i,j}$, исходя из построения алгоритма 1, может быть получено тремя способами, в зависимости от положения подматрицы, в которой оно задано. Пусть $P_{i,j}$ - ячейка матрицы P_Z. В любом из случаев P_Z вычисляется как $P_Z \cup (T_X \times T_Y)$. По определению произведения $T_X \times T_Y$, для каждого пути i, j будут получены все произведения нетерминалов из конъюнктов, выводящих все подстроки $l(\pi)$, такие,что путь $i\pi j$ разбит на две части вершиной k, такой, что $i < k < j$. По индукционному предположению, т.е. если $(i, k) \in R_B$, то $B \in T_{i,k}$ и если $(k, j) \in R_C$, то $C \in T_{k,j}$. Тогда произведение матриц $T_X \times T_Y$ даст все возможные пары нетерминалов $\{(BC)\} \in P_{i,j}$ для всех подстрок. После применения функции f (подбора

соответствующего правила грамматики), $T_{i,j}$ будет содержать нетерминалы, полученные после применения правил. Тогда, исходя из правил грамматики, получим, что если $(i,j) \in R_A$, то $A \in T_{i,j}$ (объединяем существующие поддеревья для подстрок в одно дерево). А это значит, что утверждение верно для дерева разбора высотой $h + 1$.

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

5. Пример работы алгоритма

6. Выводы

Список литературы

[1]. Warchał, Ł., & Streszczenie (2012). Using Neo4j graph database in social network analysis.

[2]. Sridharan, M., Gopan, D., Shan, L., Bod´ık, R.: Demand-driven points-to analysis for Java. In: Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, (OOPSLA2005). (2005) 59–76

[3]. Alexander Okhotin, Boolean grammars, Information and Computation, Volume 194, Issue 1, 2004, Pages 19-48.

[4]. Thomas R. Dean, James R. Cordy, Andrew J. Malton, Kevin A. Schneider, Agile parsing in TXL, Autom. Softw. Eng. 10 (4) (October 2003) 311–336.

[5]. Andrew Stevenson and James R. Cordy. 2015. Parse views with Boolean grammars. *Sci. Comput. Program.* 97, P1 (January 2015), 59-63. DOI=http://dx.doi.org/10.1016/j.scico.2013.11.007

[6]. Alexander Okhotin, Parsing by matrix multiplication generalized to Boolean grammars, Theoretical Computer Science, Volume 516, 2014, Pages 101-120, ISSN 0304-3975, https://doi.org/10.1016/j.tcs.2013.09.011.

[7]. Azimov, R., and Grigorev, S. (2017). Graph Parsing by Matrix Multiplication. CoRR, abs/1707.01007.

[8]. Hellings. J. 2014. Conjunctive context-free path queries. In: Proc. of ICDT'14, pp.119–130.

[9]. Yuan H., Eugster P. (2009) An Efficient Algorithm for Solving the Dyck-CFL Reachability Problem on Trees. In: Castagna G. (eds) Programming Languages and Systems. ESOP 2009. Lecture Notes in Computer Science, vol 5502.

[10]. R. E. Tarjan, Depth-first search and linear graph algorithms, SIAMJ. Comput. 1 (2) (1972) 146–160.

[11]. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., & Zhu, Y. (2003). Bounded model checking. Advances in Computers, 58, 117-148.