

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

Аннотация.

Ключевые слова: синтаксический анализ графов; булевы грамматики; матричные операции; ациклический граф; топологическая сортировка; деревья; 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. Часто используемыми в статическом анализе кода графами являются деревья, например, абстрактное синтаксическое дерево (AST), дерево вызовов с контекстами (CCT, calling context tree) [2], граф потока управления в технике анализа Bounded model checking [11] и другие.

Обычно графовые данные имеют большой объем, поэтому важно обрабатывать их с помощью запросов. Так как роль графов – представить отношения между различными объектами, результатом запроса будет отношение между объектами (вершинами графа). Естественным способом описать отношения между графовыми объектами является использование формальной грамматики над метками рёбер графа. Наиболее популярными являются запросы, которые используют контекстно-свободные (КС) грамматики. Существуют булевы грамматики, расширяющие класс КС-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 решение 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], решающая задачу 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 небольшом 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 $G = (N, \Sigma, P, S)$, мы обозначим отношения $R_A \subseteq V \times V$ для каждого $A \in N$ следующим образом:

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

где πm – 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, m, n) , для которых путь $m \pi n$ 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:

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

Тогда правила булевой грамматики можно представить в виде:

$$A \rightarrow \alpha_1 \wedge \alpha_2 \wedge \dots \wedge \alpha_m \wedge \neg \beta_1 \wedge \neg \beta_2 \wedge \dots \wedge \neg \beta_n,$$

где A – нетерминал, $m + n \geq 1$, $\alpha_1, \alpha_2 \dots \alpha_m, \beta_1, \beta_2 \dots \beta_n$ – строки, порождённые алфавитами Σ и N .

3. Существующие решения

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 (деревом), 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 КС 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 решения задачи синтаксического анализа ациклических графов и показана его корректность.

Пусть n - число вершин в графе D , вершины пронумерованы. Алгоритм использует следующие структуры данных:

1) Таблица парсинга T размером $n \times n$ представляющая собой верхнетреугольную матрицу, где для каждого элемента T_{ij} выполняется соотношение:

$$T_{ij} = \{A \mid a_{i+1}, \dots, a_j \in L(G_A)\},$$

где $L(G_A)$ – язык, порожденный грамматикой G и $0 \leq i < j \leq n$, i, \dots, j - номера вершин графа.

Иными словами, T_{ij} есть результирующий набор отношений

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

для всех $A \in N$ и фиксированных i, j .

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:

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

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

$$P_{i,j} = \{(B, C) \mid a_{i+1}, \dots, a_j \in L(G_B), L(G_C)\},$$

для всех $B, C \in N$ и $0 \leq i < j \leq n$.

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

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

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

$$f(P) = \{A \mid \exists A \rightarrow B_1 C_1 \wedge \dots \wedge B_m C_m \wedge \neg D_1 E_1 \wedge \dots \wedge \neg D_{m'} E_{m'} \\ \in R: (B_t, C_t) \in P \text{ и } (D_t, E_t) \notin P \text{ для } \forall t\}$$

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

Алгоритм состоит из двух рекурсивных процедур:

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

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

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

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

Полный псевдокод алгоритма приведен в листинге 1.

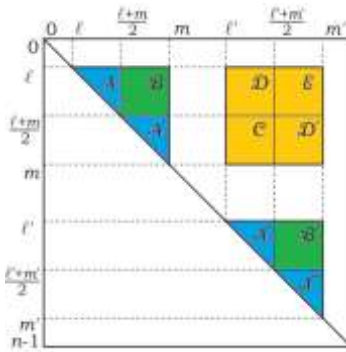


Рис. 1. Расположение подматриц для расчёта таблиц T и P .

Fig. 1. Submatrices for calculating T and P tables

```

/* D – помеченный граф с пронумерованными вершинами
/* 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) ∈ E
10:   Ti,j ← {A | A → a ∈ R}
11: compute(0, n - 1)

12: compute(l, m):
13: if m - l ≥ 4 then
14:   compute(l,  $\frac{l+m}{2}$ )
15:   compute( $\frac{l+m}{2}$ , m)
16: complete(l,  $\frac{l+m}{2}$ ,  $\frac{l+m}{2}$ , m)

17: complete(l, m, l', m')
18: if (l, m, a) ∈ E and m < l' then
19:   Tl,l' ← f(Pl,l')
20: else if m - l > 1 then
21:   B ← (l,  $\frac{l+m}{2}$ ,  $\frac{l'+m'}{2}$ , m')
22:   B' ← ( $\frac{l+m}{2}$ , m, l',  $\frac{l'+m'}{2}$ )
23:   C ← ( $\frac{l+m}{2}$ , m, l',  $\frac{l'+m'}{2}$ )
24:   D ← (l,  $\frac{l+m}{2}$ , l',  $\frac{l'+m'}{2}$ )
25:   D' ← ( $\frac{l+m}{2}$ , m,  $\frac{l'+m'}{2}$ , m')
26:   E ← (l,  $\frac{l+m}{2}$ ,  $\frac{l'+m'}{2}$ , m')
27:   complete(C)
28:   PD ← PD ∪ (TB × TC)
29:   complete(D)
30:   PD' ← PD' ∪ (TC × TB')
31:   complete(D')
32:   PE ← PE ∪ (TB × TD')
33:   PE ← PE ∪ (TD × TB')
34:   complete(E)

```

Листинг. 1. Алгоритм синтаксического анализа ациклических графов.

Listing. 1. DAG-parsing algorithm

5. Время работы и корректность алгоритма

Корректность алгоритма следует из выполнимости следующих утверждений.

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

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.

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

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

Индукционный переход. Предположим, утверждение верно для всех деревьев разбора высотой n . Докажем, что теорема верна для деревьев разбора высотой $n+1$.

Рассмотрим дерево разбора для $l(\pi)$ высотой $n+1$. Так как грамматика в нормальной форме, то у данного дерева будут поддеревья, выводящие подстроки $l(\pi)$ (возможно пересекающиеся). По свойству топологической сортировки для всех индексов k, l этих подстрок (начала и конца соответствующих путей) выполняется неравенство $i \leq k < m \leq j$. По Лемме 1 вызов $compute(i, j)$ посчитает $T_{i,j} = \{A \mid a_{i+1}, \dots, a_j \in L(G_A)\}$ для всех $i \leq k$

$< l < j$. По индукционному предположению, для всех подпутей $k\pi'm$ пути $i\pi j$ и нетерминалов $A \in T_{k,m}$ верно, что $l(\pi') \in L(G_A)$. $T_{i,j}$ вычисляется как $f(P_{i,j})$. $P_{i,j}$ по алгоритму 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, выводющих все подстроки $l(\pi)$, такие, что путь $i\pi j$ разбит на две части вершиной k , такой что $i < k < j$. По индукционному предположению и по Лемме 1 для всех таких подстрок будут существовать деревья разбора, и высота их не будет превышать n . Тогда исходя из правил грамматики и того, что она в нормальной форме, произведение матриц $T_X \times T_Y$ даст все возможные нетерминалы $A \in T_{i,j}$, такие, что $l(\pi) \in L(G_A)$. А это значит, что утверждение верно для дерева разбора высотой $n + 1$.

Время работы алгоритма аналогично времени работы алгоритма Охотина [6] и составляет $O(|G| \cdot BMM(n) \log n)$, где $|G|$ - размер входной булевой грамматики, n - число вершин в графе D , $BMM(n)$ - время умножения булевых матриц размера $n \times n$.

6. Пример работы алгоритма

7. Выводы

Список литературы

- [1]. Warchał, Ł., & Streszczenie (2012). Using Neo4j graph database in social network analysis.
- [2]. Sridharan, M., Gopan, D., Shan, L., Bodík, 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.