Generalized LL parsing for context-free constrained path search problem

Semyon Grigorev
Saint Petersburg State University
7/9 Universitetskaya nab.
St. Petersburg, 199034 Russia
semen.grigorev@jetbrains.com

Anastasiya Ragozina
Saint Petersburg State University
7/9 Universitetskaya nab.
St. Petersburg, 199034 Russia
ragozina.anastasiya@gmail.com

ABSTRACT

Aaaabstract is very abstract.... word1 word2 word3 word4 word5 word6 word7 word8 word9 word10 word1 word2 word3 word4 word5 word6 word7 word8 word9 word10 word1 word2 word3 word4 word5 word6 word7 word8 word9 word10 word1 word2 word3 word4 word5 word6 word7 word8 word9 word10 word1 word2 word3 word4 word5 word6 word7 word8 word9 word10 word1 word2 word3 word4 word5 word6 word7 word8 word9 word10 word1 word2 word3 word4 word5 word6 word7 word8 word9 word10 word1 word2 word3 word4 word5 word6 word7 word8 word9 word10 word1 word2 word3 word4 word5 word6 word7 word8 word9 word10 word1 word2 word3 word4 word5 word6 word7 word8 word9 word10

1. INTRODUCTION

Graph data model and graph data bases are very popular in many different areas such as bioinformatic, semantic web, social networks etc. Extraction of paths satisfying specific constraints may be useful for graph structured data investigation and for relations between data items detection. Path querying with constrains formulated in terms of formal grammars is a specific problem named formal language constrained path problem [3] and research in this area is still actual [8].

Classical parsing techniques can be used to solve formal language constrained path problem thus we can use "graph parsing" and it may be required not only in graph data base quering but also in other different areas: formal verification, and string-embedded language processing, for example.

There is set of solutions in DB area but !!! Query result exploration is a challenge [6]. Classical parsing allow to construct derivation tree which is structural representation of parsed sentence. Structural represenatation of query result can be useful for its exploration and qury debugging. We propose graph parsing technique which allow to construct structural representation of query result with reletion to grammar query or derivation of result.

Graph parsing can be also used in string-embedde languages prcessing. Regular approximation for value set of string variable can by represened as directed graph of related fite automata. In orded to check corectness or safety (sql injections)... all generated strings (all paths from start states to final states) are correct w.r.t some context-free grammar. For example grammar of one of SQL dialects. GLR-based for string-embedded SQL checking [2, 4]. Solution based on RNGLR [11] for relaxed parsing of string-embedded languages [19] which allow to find all path between two specified vertices.

Classical solution in DB area use such parsing algorithms as CYK or Earley. In string-embedded languages analysis (RN)GLR is used. It has better time complexity.

2. GENERALIZED LL PARSING ALGORITHM

Generalized LL (GLL) is generalized top-down parsing algorithm which handle all context-free grammars (including left recursive) with worst-case cubic time complexity and linear for LL grammars. GLL is native for grammar, can be simple created blah-blah-blah.

GLL use descriptors for parsing states specification. Each descriptor contains full specification of process state anought to start parsing from state stored.

Descriptor is a triple (L, s, j) where L is a line label, s is a stack and j is a position in the input.

allows to restore parsing

Graph structured stack (GSS) [17] for multiple stack combining to prevent duplication. In GLL each GSS node is pair of position in input and grammmar slot. Grammar slot is a !!!

2.1 Shared packed parse forest

Shared Packed Parse Forest (SPPF) is a spetial data structure for derivation forest compact representation which allow to reuse common nodes and subtrees. As a result multiple derivation trees, which csn be produced in case of ambiguos grammar, can be compressed in one SPPF with optimal reusing of common parts. Binarized form of SPPF proposed in [15] and it allow to achive worst-case cubic space complexity. GLL can use SPPF [13] for results representation achive cubic space complexity with binarised version.

Let we present an example of SPPF for ambiguos grammar G_0 (pic 1).

```
0: s = NUM
1: s = LBR s RBR
2: s = s s
```

Figure 1: Grammar G_0

Here `N` is token for number, `L` and `R` are tokens for '(' and ')' respectively.

Let we parse the sentence `(1)(2)(3)`. There are two diferent lefmost derivations of this sentence in grammar G_0 ($\rightarrow^n$ denote an application of production with nimber n):

1. $s \rightarrow^2 ss \rightarrow^2 sss \rightarrow^1 LsRss \rightarrow^0 LNRss \rightarrow^1 LNRLsRs \rightarrow^1 LNRLsRs \rightarrow^0 LNRLNRs \rightarrow^1 LNRLNRLsR \rightarrow^0 LNRLNRLNR$

2. $s \rightarrow^2 ss \rightarrow^1 LsRs \rightarrow^0 LNRs \rightarrow^2 LNRss \rightarrow^1 LNRLsRs \rightarrow^1 LNRLsRs \rightarrow^0 LNRLNRs \rightarrow^1 LNRLNRLsR \rightarrow^0 LNRLNRLNR$

As far as there are tho different derivations, SPPF should contains 2 different trees and it is presented in figure 2: result SPPF 2a and trees for derivation 1 2b and derivation 2 2b respectively.

Binarised SPPF is a graph where !!! and each node has one of four types and one node marked as 'root' — node for start nonterminal.

- terminal node
- nonterminal node
- intermidiate node
- packed node

Further we will remove redudant intermidiate and packed nodes from SPPF to simplify it and decrease size of structure.

3. PRELIMINARIES

Let we introduce some definitions.

- Context-free grammar $G = (N, \Sigma, P, S)$ where N is a set of nonterminal symbols, Σ is a set of nonterminal symbols, $S \in N$ is a start nionterminal, and P is a productions set.
- Directed graph $M = (V, E, L)$ where V — vertices set, $L \subseteq \Sigma$ — edge labels set, $E \subseteq V \times L \times V$. We assume that there are no parallel edges with equal labebs: for every $e_1 = (v_1, l_1, v_2) \in E, e_2 = (u_1, l_2, u_2) \in E$ if $v_1 = u_1$ and $v_2 = u_2$ then $l_1 \neq l_2$.
- Helper function for edge's tag calculation $tag : E \rightarrow L; tag(e = (v_1, l, v_2), e \in E) = l$.
- Concatenation operation $\oplus : L^+ \times L^+ \rightarrow L^+$.
- Path p in graph M.
$p = (v_0, l_0, v_1), (v_1, l_1, v_2), \dots, (v_{n-1}, l_{n-1}, v_n) = e_0, e_1, \dots, e_{n-1}$ where $v_i \in V, e_i \in E$, $l_i \in L$, $|p| = n \geq 1$.
- Set of paths $P = \{p : p \text{ path in } M\}$
- Helper function for string produced by path calculation $\Omega : P \rightarrow L^+$.
$\Omega(p = e_0, e_1, \dots, e_{n-1}, p \in P) = tag(e_0) \oplus \dots \oplus tag(e_{n-1})$.

As a result we can define that context-free language constrained path querying meens that each path $p = e_0, \dots, e_{n-1}$ from result set satisfied with next constraint: $\Omega(p) \in L(G)$.

As a motivation of context-free constraints importance let we introduce the next example. Let we have graph $M = (\{0; 1; 2; 3\}, E, \{A; B\})$ presented in figure 3 where labels represent $parent(A)$ and $child(B)$ relations. Suppose for each $n \leq 1$ we want to find all n-th generation descendants with a common ancestor. In the other worlds, we wath to find all paths p, such that $\Omega(p) \in \{AB; AABB; AAABBB; \dots\}$ or $\Omega(p) = A^n B^n$ where $n \geq 1$. This constraint can not be specified with regular language as far as $L = \{A^n B^n; n \geq 1\}$ is not regular but context free. Required language can be specified by grammar G_1 presented in picture 4 where $N = \{s; middle\}$, $\Sigma = \{A; B\}$, and $S = s$.

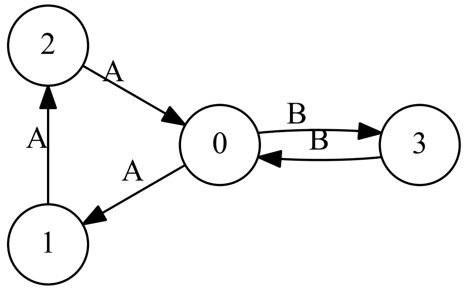


Figure 3: Input graph M

```
0: s = L s R
1: s = middle
2: middle = L R
```

Figure 4: Grammar G_1 for language $L = \{L^n R^n; n \geq 1\}$

4. GLL-BASED GRAPH PARSING

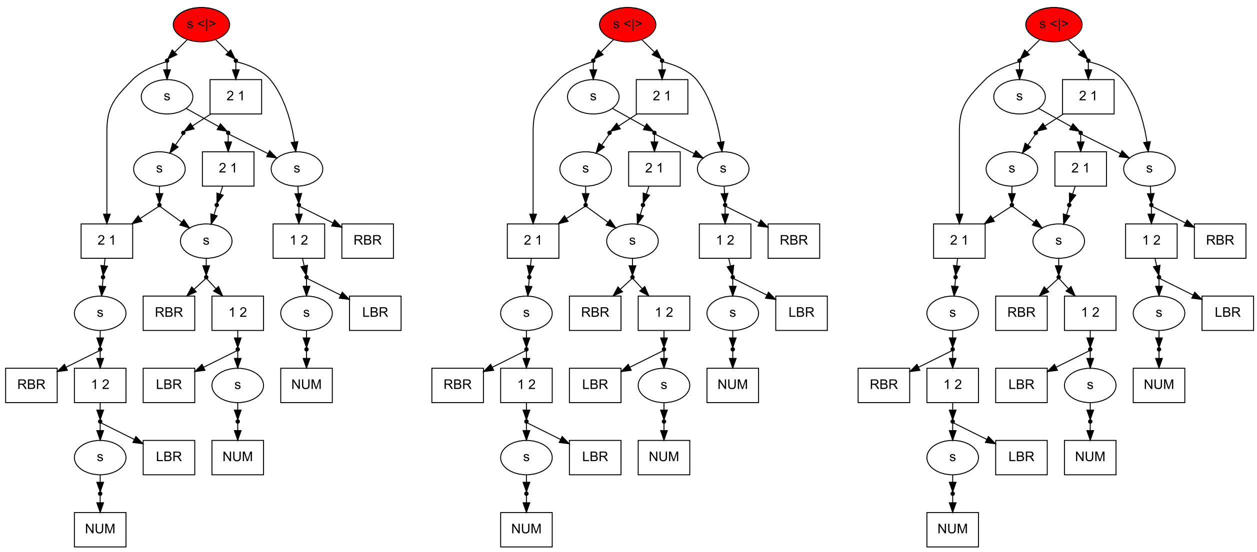
We propose a context-free language constrained path problem solution which allow to find all paths satisfied specified arbitrary context-free grammar and to construct implicit representation of result. Finite representation of result set with structure related to specified grammar may be useful not only for results understanding and processing but also for query debugging especially for complex queries.

Our solution is based on generalized LL (GLL) [12, 1] parsing algorithm which allow to process ambiguous context-free grammars. Complexity is $O(n^3)$ in worst case and linear for unambiguous grammars, that better then complexity of CYK and Earley which used as base in other solutions (for example [5], [16]). This fact allow to demonstarte better performance on linear subgraphs and unambiguous grammars. Also it is not necessary to transform input grammar to CNF which required for CYK.

In order to use GLL for graph parsing we need only use graph verticea as position in input. As far as we work with context-free languages it is not important how this descriptor was created, and so descriptors management and other basic mechanisms of original algorithm can be reused "as is". We can merge it if thea are equal.

We implement some optimizations: [1]

We also use binarised SPPF for result representation whish allow to simplify query debugging and result explortion. In our case more then one root may be specified. For example, look at picture!!!! We



(a) SPPF (b) Tree for derivation 1 (c) Tree for derivation 2

Figure 2: SPPF for sentence `(1)(2)(3)` and grammar G_0

4.1 Complexity

Time complexity estimation in terms of input graph and grammar size is pretty similar to estimation of GLL complexity explained in [7].

Descriptor is a triple (L, s, j) where $|L| = f(G)$ — some grammar-depended constant, $|j| = |V|$, and $|s| = |GSS.Nodes|$. GSS node N is a pair (lbl, j) where $|lbl| = f(G)$ and $|j| = |V|$. Thus we can create at most V^2 descriptors. For each descriptor we should examine all outgoing edges which can be bounded with $\max_{v \in V}(deg^+(v))$ where $deg^+(v)$ is outdegree of vertex v. Thus we need $O(|V|^2 * (\max_{v \in V}(deg^+(v))))$ operation for outgoing edges processing.

For all elsements produced in previous step we should perform search in internal data structures. It is possible in linear (w.r.t position count, $|V|$ in our case) time [7]. So, worst-case complexity of proposed algorithm is

$$O(V^3 * \max_{v \in V}(deg^+(v))) \tag{1}$$

To get averege-case complexity we can calculate averege outdegree:

$$O\left(|V|^3 * \frac{\sum_{v \in V} deg^+(v)}{|V|}\right) = \tag{2}$$

$$O\left(|V|^2 * \sum_{v \in V} deg^+(v)\right) = \tag{3}$$

$$O\left(|V|^2 * |E|\right) \tag{4}$$

For unambiguous grammar complexity should be linear in terms of vertices count for the same reason as in classical GLL.

From 1 and 2 we can get estimations for linear input: for any $v \in V deg^+(v) \leq 1$, so $\max_{v \in V}(deg^+(v)) = 1$ and from 1 we get $O(|V|^3)$. By the same way we can get $O(|V|^3)$ form 2 denote that $|E| = |V| - 1$ in case of linera input.

As discussed in [7] achiving of theoretical complexity reqired special datastructures which can be irrational for practice implementation and it is necessary to finde balance between performance, software complexity, and hardware resources. So in practice we can get slightly worse performance than theoretical estimation.

4.2 Example

In details, main function input is graph M, set of start vertices $V_s \subseteq V$, set of final vertices $V_f \subseteq V$, grammar G_1. Output is Shared Packed Parse Forest (SPPF) [10] — finite data structure which contains all derivation trees for all paths in M, $\Omega(p) \in L(G_1)$ and allows to reconstruct any of paths implicitly. As far as we can specify sets of start and final vertices, our solution can find all paths in graph, all paths from specified vertex, all paths between specified vertices. Also SPPF represents a structure of paths in terms of derivation which allow to get more useful information about result.

Let we introduce the next example. Grammar G_1 is a query and we want to find all paths in graph M (presented in picture 3) matched this query. Result SPPF for this input is presented in picture 5. Note that presented version does not contein obsolete nodes.

We use next markers for nodes which similar to original SPPF but have some additional information in order to relation with graph.

- Node with rectangle shape labeled with (v_0, T, v_1) is terminal node. Each terminal node corresponds with edge in the input graph: for each node with label (v_0, T, v_1) there is $e \in E : e = (v_0, T, v_1)$. Duplication of terminal nodes is only for figure simplification.
- Node with oval shape labeled with (v_0, nt, v_1) is nonterminal node. This node denote that there is at least one path p from vertex v_0 to vertex v_1 in input graph

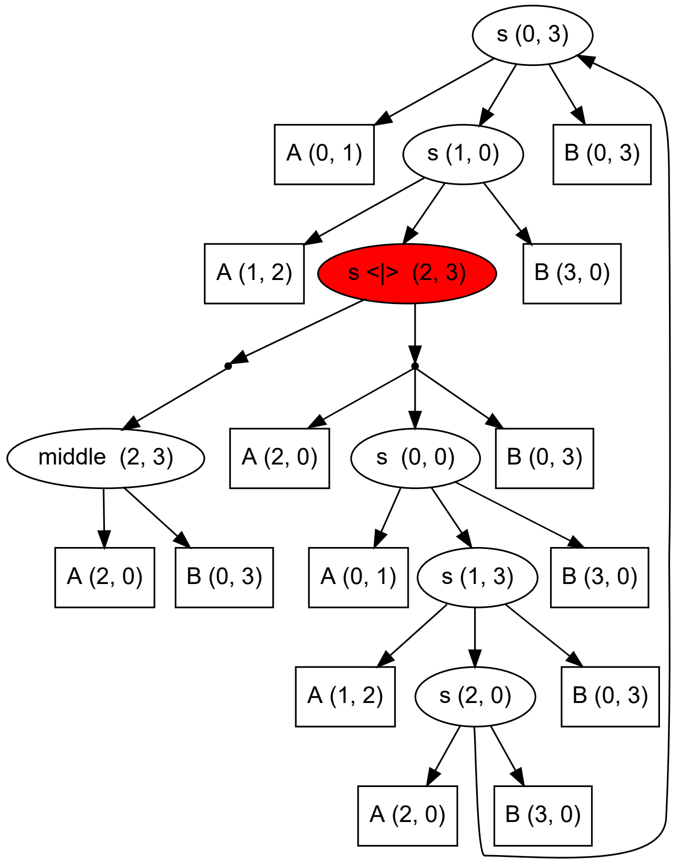


Figure 5: Result SPPF for input graph M(pic. 3) and query G_1(pic. 4)

M such that $nt \Rightarrow_G^* \Omega(p)$. All paths matched this condition can be extracted from SPPF by left-to-right top-down graph traversal started from respective node.

- Filled node with oval shape labeled with $(<|>(v_0, nt, v_1))$ is nonterminal node denote that there are more then one path from v_0 to v_1 such that $nt \Rightarrow_G^* \Omega(p)$.
- Node with dot shape is used for representation of derivation variants. Subgraph with root in one such node is one variant of derivation. Parent of such nodes is always node with label $(\lhd\!\rhd\ (v_0, nt, v_1))$.
- v_0 and v_1 are left and right extensions of node respectively.

As an example of derivation structure usage we can find 'middle' of any path in example above simply by finding corresponded nonterminal *middle* in SPPF. So we can found that there is only one common ancestor for all results and it is vertex with $id = 0$.

Extensions stored in nodes allow to check whether path from u to v exists and extract it. Path extraction is SPPF traversal. Let for example we want to find path satisfying specified constraints fron vertex 0. To do this we should find vertices with label $(0, s, \_)$ in SPPF. There are two vertices: $(0, s, 0)$ and $(0, s, 3)$. In our example there is cycle in SPPF so there are **at least** two different paths: $p_0 = \{(0, A, 1); (1, A, 2); (2, A, 0); (0, B, 3); (3, B, 0); (0, B, 3)\}$ and $p_1 = \{(0, A, 1); (1, A, 2); (2, A, 0); (0, A, 1); (1, A, 2); (2, A, 0); (0, B, 3); (3, B, 0); (0, B, 3); (3, B, 0); (0, B, 3); (3, B, 0)\}$.

5. EVALUATION

We perform some experiments on syntatic graphs. Full graphs and graphs with structure presented in figure !!!. All paths from all vertices and all paths from one specified vertex. For full graph also all paths between two specified vertices.

We use two grammars for balanced brakets in order to investigate performance relations with grammar ambiguity: ambiguos grammar G_0 1 and unambiguos grammar G_2 6.

```
0: s = L s R s
1: s = eps
```

Figure 6: Unambiguos grammar G_2 for balanced brackets

All tests were performed on a PC with following characteristics:

- OS Name: Microsoft Windows 10 Pro
- System Type: x64-based PC
- CPU: Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz, 3601 Mhz, 4 Core(s), 4 Logical Processor(s)
- RAM: 32 GB

Results presented in figure 7. From all and from one vertex is because descriptors reusing.

set terminal latex set output 'ContextFreeConstrainedPathFindingIn

Figure 7: Performance on C graph for grmmars G_0 and G_2

To summarise we can say that performance for unambiguos grammars is better then for ambiguos.

6. CONCLUSION AND FUTURE WORK

We propose GLL-based algorithm for context-free path querying which construct finite structural representation of all paths satisfying given constraint. Provided data structure can be useful for result investigation and processing, and query debugging. Presented algorithm implemented in F# and available on GitHub:https://github.com/YaccConstructor/YaccConstructor.

In order to estimate practical !!! we should perform evaluation on real dataset and real queries.

Also we are working on performance improvement by implementation of recently proposed modifications in original GLL algorithm [14]. One of direction of our research is generalization of grammar factorization proposed in [14] which may be useful for regular query processing.

We are working on utilisation of GPGPU and multicore CPU power for graph parsing problem with Valiant [18] algorithm modification proposed by Alexander Okhotin [9]. One of possible benefit is ability to process more expressive queries because modification proposed by Alexander Okhotin extended to support boolean grammars.

7. REFERENCES

[1] A. Afroozeh and A. Izmaylova. Faster, practical gll parsing. In *International Conference on Compiler Construction*, pages 89–108. Springer, 2015.

[2] A. Annamaa, A. Breslav, J. Kabanov, and V. Vene. An interactive tool for analyzing embedded sql queries. In *Asian Symposium on Programming Languages and Systems*, pages 131–138. Springer, 2010.

[3] C. Barrett, R. Jacob, and M. Marathe. Formal-language-constrained path problems. *SIAM Journal on Computing*, 30(3):809–837, 2000.

[4] A. Breslav, A. Annamaa, and V. Vene. Using abstract lexical analysis and parsing to detect errors in string-embedded dsl statements. In *Proceedings of the 22nd Nordic Workshop on Programming Theory*, pages 20–22, 2010.

[5] J. Hellings. Conjunctive context-free path queries. 2014.

[6] P. Hofman and W. Martens. Separability by short subsequences and subwords. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 31. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[7] A. Johnstone and E. Scott. Modelling gll parser implementations. In *International Conference on Software Language Engineering*, pages 42–61. Springer Berlin Heidelberg, 2010.

[8] J. A. Miller, L. Ramaswamy, K. J. Kochut, and A. Fard. Research directions for big data graph analytics. In *2015 IEEE International Congress on Big Data*, pages 785–794. IEEE, 2015.

[9] A. Okhotin. Parsing by matrix multiplication generalized to boolean grammars. *Theoretical Computer Science*, 516:101–120, 2014.

[10] J. G. Rekers. *Parser generation for interactive environments*. PhD thesis, Citeseer, 1992.

[11] E. Scott and A. Johnstone. Right nulled glr parsers. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 28(4):577–618, 2006.

[12] E. Scott and A. Johnstone. Gll parsing. *Electronic Notes in Theoretical Computer Science*, 253(7):177–189, 2010.

[13] E. Scott and A. Johnstone. Gll parse-tree generation. *Science of Computer Programming*, 78(10):1828–1844, 2013.

[14] E. Scott and A. Johnstone. Structuring the gll parsing algorithm for performance. *Science of Computer Programming*, 125:1–22, 2016.

[15] E. Scott, A. Johnstone, and R. Economopoulos. Brnglr: a cubic tomita-style glr parsing algorithm. *Acta informatica*, 44(6):427–461, 2007.

[16] P. Sevon and L. Eronen. Subgraph queries by context-free grammars. *Journal of Integrative Bioinformatics*, 5(2):100, 2008.

[17] M. Tomita. An efficient context-free parsing algorithm for natural languages. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'85, pages 756–764, San Francisco, CA, USA, 1985. Morgan Kaufmann Publishers Inc.

[18] L. G. Valiant. General context-free recognition in less than cubic time. *Journal of computer and system sciences*, 10(2):308–315, 1975.

[19] E. Verbitskaia, S. Grigorev, and D. Avdyukhin. Relaxed parsing of regular approximations of string-embedded languages. In *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, pages 291–302. Springer International Publishing, 2015.