Rytter-style Algorithm for Context-Fre Path Querying

Semyon Grigorev
Saint Petersburg State University
St. Petersburg, Russia
semen.grigorev@jetbrains.com

Ekaterina Shemetova
Saint Petersburg State University
St. Petersburg, Russia
katyacyfra@gmail.com

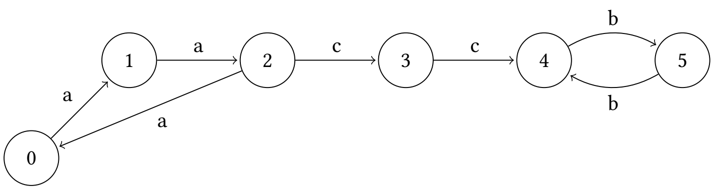


Figure 1: The input graph

ACM Reference Format:

Semyon Grigorev and Ekaterina Shemetova. 2019. Rytter-style Algorithm for Context-Fre Path Querying. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The problem is to check the emptyness of the intersection of the regular language R mhich is represented as FSA A with number of states n, and context-free language L in less then $O(n^3)$. The equvalent problem is a context-free reachability problem [**?**].

First step is a reduction of the given problem to BMM(n) (with possibly polylogarithmic factors). We hope that such reduction helps us to get algorithm for CFL reachability with $\widetilde{O}(BMM(n))$ time complexity where $\widetilde{O}$ meens polylog factors.

2 FROM ARBITRARY CFPQ TO DYCK QUERY

This reduction is inspired by the construction described in [2].

Consider a context-free grammar $\mathcal{G} = (\Sigma, N, P, S)$ in BNF where Σ is a terminal alphabet, N is a nonterminal alphabet, P is a set of productions, $S \in N$ is a start nonterminal. Also we denote a directed labeled graph by $G = (V, E, L)$ where $E \subseteq V \times L \times V$ and $L \subseteq \Sigma$.

We should construct new input graph G' and new grammar $\mathcal{G}'$ such that $\mathcal{G}'$ specifies a Dyck language and there is a simple mapping from CFPQ($\mathcal{G}', G'$) to CFPQ($\mathcal{G}, G$). Step-by-step example with description is provided below.

Let the input grammar is

$$S \rightarrow a\ S\ b \mid a\ C\ b$$
$$C \rightarrow c \mid C\ c$$

The input graph is presented in fig. 1.

(1) Let $\Sigma_{()} = \{t_(, t_) | t \in \Sigma\}$.

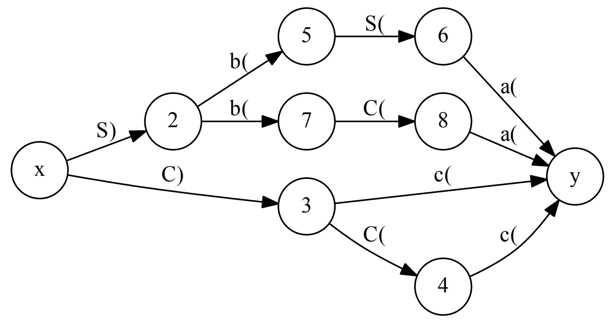


Figure 2: The $M_{\mathcal{G}}$ graph

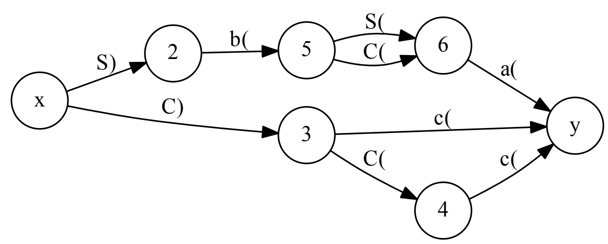


Figure 3: The minimized $M_{\mathcal{G}}$

(2) Let $N_{()} = \{N_(, N_) | N \in N\}$.
(3) Let $M_{\mathcal{G}} = (V_{\mathcal{G}}, E_{\mathcal{G}}, L_{\mathcal{G}})$ is a directed labeled graph, where $L_{\mathcal{G}} \subseteq (\Sigma_{()} \cup N_{()})$. This graph is created the same manner as described in [2] but we do not require the grammar be in CNF. Let $x \in V_{\mathcal{G}}$ and $y \in V_{\mathcal{G}}$ is "start" and "final" vertices respectively. This graph may be treated as a finite automaton, so it can be minimized and we can compute an ε-closure if the input grammar contains ε productions. The graph $M_{\mathcal{G}}$ for our example is presented in fig. 2. The minimized graph is presented in fig. 3.
(4) For each $v \in V$ create $M^v_{\mathcal{G}}$: unique instance of $M_{\mathcal{G}}$.
(5) New graph $G^{'}$ is a graph G where each label t is replaced with $t^i_)$ and some additional edges are created:
   - Add an edge $(v', S_(, v)$ for each $v \in V$.
   - And the respective $M^v_{\mathcal{G}}$ for each $v \in V$:
     – reattach all edges outgoing from x^v ("start" vertex of $M^v_{\mathcal{G}}$) to v;
     – reattach all edges incoming to y^v ("final" vertex of $M^v_{\mathcal{G}}$) to v.

   New input graph is ready. It is presented in fig. 4.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

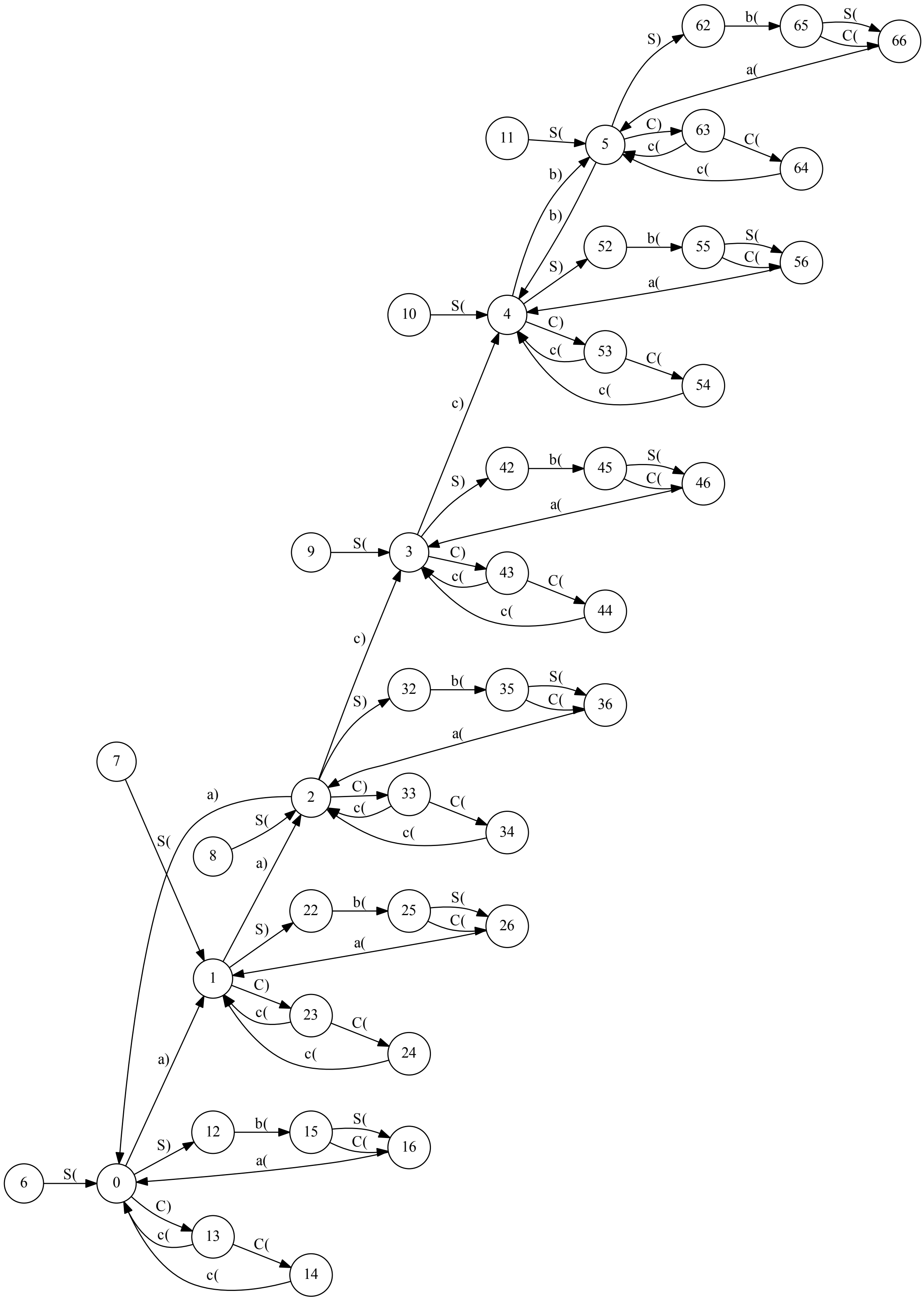


Figure 4: New input graph

(6) New grammar $\mathcal{G}' = (\Sigma', N', P', S')$ where $\Sigma' = \Sigma_{()} \cup N_{()}$, $N' = \{S'\}$, $P' = \{S' \to b_(\ S' \ b_); S' \to b_(\ b_) \mid b_(, b_) \in \Sigma'\} \cup \{S' \to S' \ S'\}$ is a set of productions, $S' \in N'$ is a start nonterminal.

Now, if CFPQ($\mathcal{G}', G'$) contains a pair (u'_0, v') such that $e = (u'_0, S_(, u'_1) \in E'$ is an extension edge (step 5, first subitem), then $(u'_1, v') \in$ CFPQ($\mathcal{G}, G$). In our example, we can find the following path: $7 \xrightarrow{S_(} 1 \xrightarrow{S_)} 22 \xrightarrow{b_(} 25 \xrightarrow{C_(} 26 \xrightarrow{a_(} 1 \xrightarrow{a_)} 2 \xrightarrow{C_)} 33 \xrightarrow{C_(} 34 \xrightarrow{c_(} 2 \xrightarrow{c_)} 3 \xrightarrow{C_)} 43 \xrightarrow{c_(} 3 \xrightarrow{c_)} 4 \xrightarrow{b_)} 5$. Edge $7 \xrightarrow{S_(} 1$ is the extension, so (1,5) should be in CFPQ($\mathcal{G}, G$) and it is true.

3 STRONGLY CONNECTED COMPONENTS HANDLING

Steps:

(1) Convert input graph to graph for 2-Dyck querying.
(2) Convert graph to one strongly connected component by adding edges with new unique label from synks to sources.
(3) Convert 2-Dyck grammar to grammar which can accept arbitrary path with 2-Dyck subpaths.
(4) Execute modified Rytter for one arbitrary selected vertex and its output edge.

In strongly connected components each vetex is reachcbel from another, but not each path should match required constraints. The idea is to extend grammar by the such way, that it accepts arbitrary path and provide information about parts which satisfie to original constraints. As far as we can reduce any CFPQ to 2-Dyck query, we can fix grammar as follows.

$$
\begin{aligned}
S &\to A \ S_1 \mid C \ S_2 \mid S \ S \mid A \ B \mid C \ D \\
S_1 &\to S \ B \\
S_2 &\to S \ D \\
A &\to a \\
B &\to b \\
C &\to c \\
D &\to d
\end{aligned}
$$

Let label of new edges which added in order to convert graph to single SCC is E. Arbitrary path consists of 2-Dyck subpaths connected by unbalanced parts. We can specify grammar for these paths.

$$
\begin{aligned}
S' \to & a \mid b \mid c \mid d \mid e \mid \\
& A \ S' \mid B \ S' \mid C \ S' \mid D \ S' \mid E \ S' \mid S' \ S' \mid \\
& A \ S_1 \mid C \ S_2 \mid S \ S \mid A \ B \mid C \ D \\
S \to & A \ S_1 \mid C \ S_2 \mid S \ S \mid A \ B \mid C \ D \\
S_1 \to & S \ B \\
S_2 \to & S \ D \\
A \to & a \\
B \to & b \\
C \to & c \\
D \to & d \\
E \to & e
\end{aligned}
$$

Now we can start processing from one arbitrary selected vertex.

Scheme of proof.

(1) Convertion to 2-Dyck path querying. Look at ection !!!.
(2) Convertion to single SCC. In worst case we should add $|V|$ outgoing edges for each vertes. So, time complexity is $O(|V|^2)$.
(3) Why can we select arbitrary edge for start? We can select arbitrarry vertex just because we handle SCC, so all other vertices should be reachcble from selected one. We should choose outgoing edge from selected vertex in order to fix source vertex in Rytter graph.
(4) Rytter

4 RYTTER ALGORITHM FOR GRAPH INPUT

Main idea is to adopt algorithm from [3] for CFPQ. It should be possible to perform adaptation for arbitrary CFPQ, but we are interested in case of Dyck queries because it should simplify complexity estimation.

We introduce an example and try to explain key steps. As far as example for graph and query introduced in the previous section is too big, we use another input data.

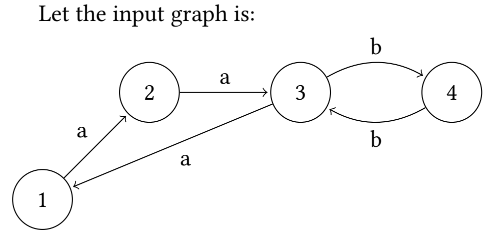
Let the input grammar is

$$
\begin{aligned}
S &\to a \ S \ b \\
S &\to a \ b
\end{aligned}
$$

The input grammar in CNF is

$$
\begin{aligned}
S &\to A \ S_1 \\
S_1 &\to S \ B \\
S &\to A \ B \\
A &\to a \\
B &\to b
\end{aligned}
$$

Let the input graph is:



We use the same notation and the semiring as proposed by Rytter in [3]. The *IMPLIED* relation for our example is presented in figure 5. Furher we will write (N_1, N_2) instead of $(N_1, i, j) \Rightarrow (N_2, k, l)$ when positions specification are not important in the context.

Initial grid graph is presented in fig 6. It can be constructed by the similar way as presented in [3] and can by stored in two $n \times n$ matrix where n is a number of vertices in input graph.

We should introduce the identity set id such that:

- $id \times A = A \times id = A$
- $id \times id = id$

This set may be constructed as follows: $id = \{(N_i, N_i) | N_i \in N\}$.

In order to compute transitive closure in logarithmic time we add self-loop with weight id to each vertex. Result is graph $\mathcal{G}$ which is presented in fig. 7.

Now we can do some observations.

- Graph $\mathcal{G}$ is pretty similar to Rytter's grid graph (except cycles which have special structure and satisfy strongly congruence restriction) and can be represented as two matrices of size $n \times n$: $\mathcal{G}_H$ and $\mathcal{G}_V$ for horizontal and vertical edges respectively. We use the same representation as Rytter. Note that self-loops should be duplicated and stored in both matrices.
- We can compute transitive closure of $\mathcal{G}_H$ and $\mathcal{G}_V$ in $\widetilde{O}(BMM(n))$ by using standard techniques fro transitive closure calculation. Let $\mathcal{G}'_H$ is a closure of $\mathcal{G}_H$ and $\mathcal{G}'_V$ is a closure of $\mathcal{G}_V$.

$$
\begin{array}{llll}
(B,2,3) \Rightarrow (S,1,3) & (B,2,4) \Rightarrow (S,1,4) & (B,2,2) \Rightarrow (S,1,2) & (B,2,1) \Rightarrow (S,1,1) \\
(B,3,4) \Rightarrow (S,2,4) & (B,3,3) \Rightarrow (S,2,3) & (B,3,2) \Rightarrow (S,2,2) & (B,3,1) \Rightarrow (S,2,1) \\
(B,1,2) \Rightarrow (S,3,2) & (B,1,3) \Rightarrow (S,3,3) & (B,1,4) \Rightarrow (S,3,4) & (B,1,1) \Rightarrow (S,3,1) \\
(S_1,2,3) \Rightarrow (S,1,3) & (S_1,2,4) \Rightarrow (S,1,4) & (S_1,2,2) \Rightarrow (S,1,2) & (S_1,2,1) \Rightarrow (S,1,1) \\
(S_1,3,4) \Rightarrow (S,2,4) & (S_1,3,3) \Rightarrow (S,2,3) & (S_1,3,2) \Rightarrow (S,2,2) & (S_1,3,1) \Rightarrow (S,2,1) \\
(S_1,1,2) \Rightarrow (S,3,2) & (S_1,1,3) \Rightarrow (S,3,3) & (S_1,1,4) \Rightarrow (S,3,4) & (S_1,1,1) \Rightarrow (S,3,1) \\
(A,2,3) \Rightarrow (S,2,4) & (A,1,3) \Rightarrow (S,1,4) & (A,3,3) \Rightarrow (S,3,4) & (A,4,3) \Rightarrow (S,4,4) \\
(A,3,4) \Rightarrow (S,3,3) & (A,4,4) \Rightarrow (S,4,3) & (A,2,4) \Rightarrow (S,2,3) & (A,1,4) \Rightarrow (S,1,3) \\
(S,2,3) \Rightarrow (S_1,2,4) & (S,1,3) \Rightarrow (S_1,1,4) & (S,3,3) \Rightarrow (S_1,3,4) & (S,4,3) \Rightarrow (S_1,4,4) \\
(S,3,4) \Rightarrow (S_1,3,3) & (S,4,4) \Rightarrow (S_1,4,3) & (S,2,4) \Rightarrow (S_1,2,3) & (S,1,4) \Rightarrow (S_1,1,3)
\end{array}
$$

Figure 5: *IMPLIED* relation for our example

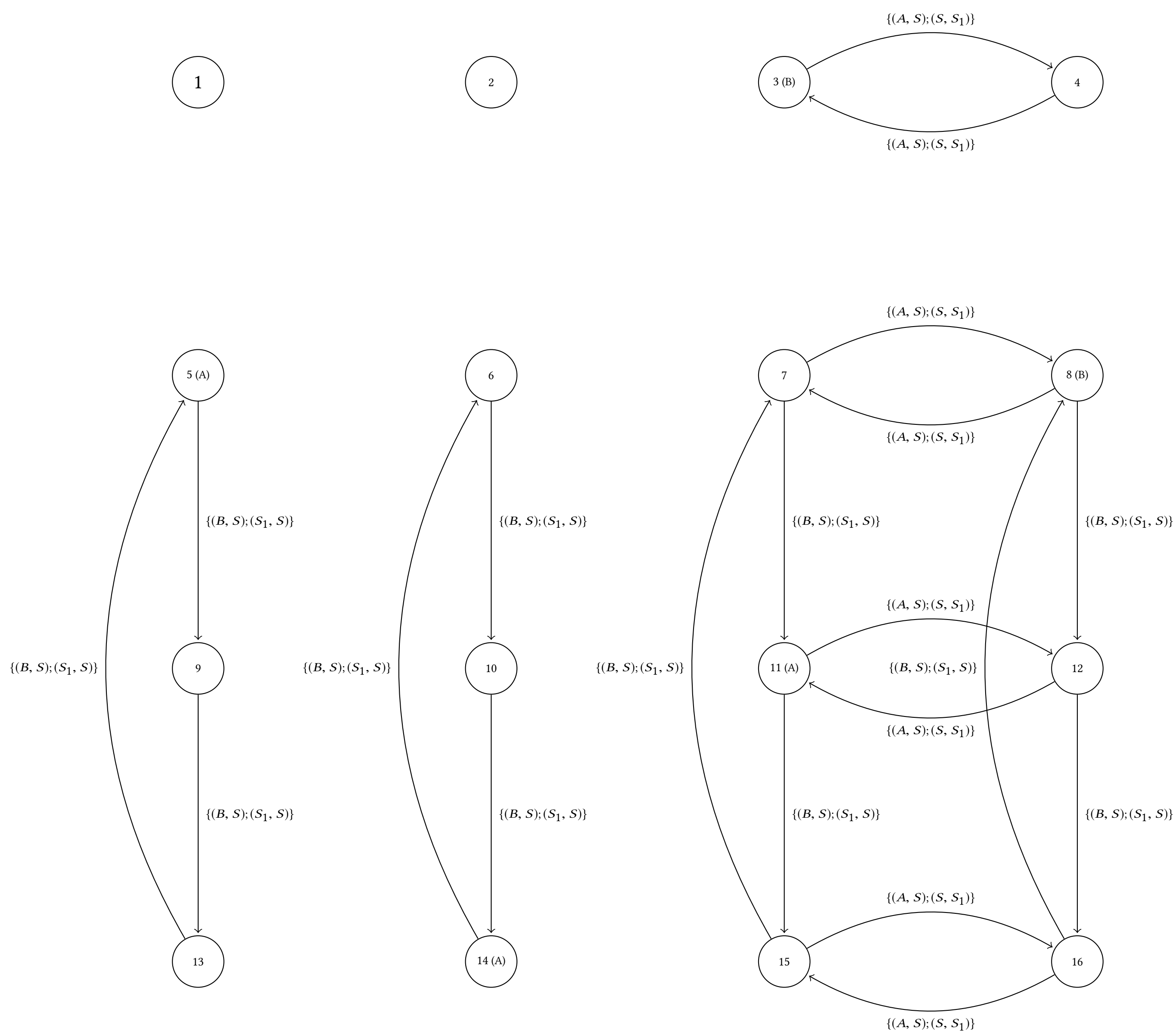


Figure 6: Initial grid graph

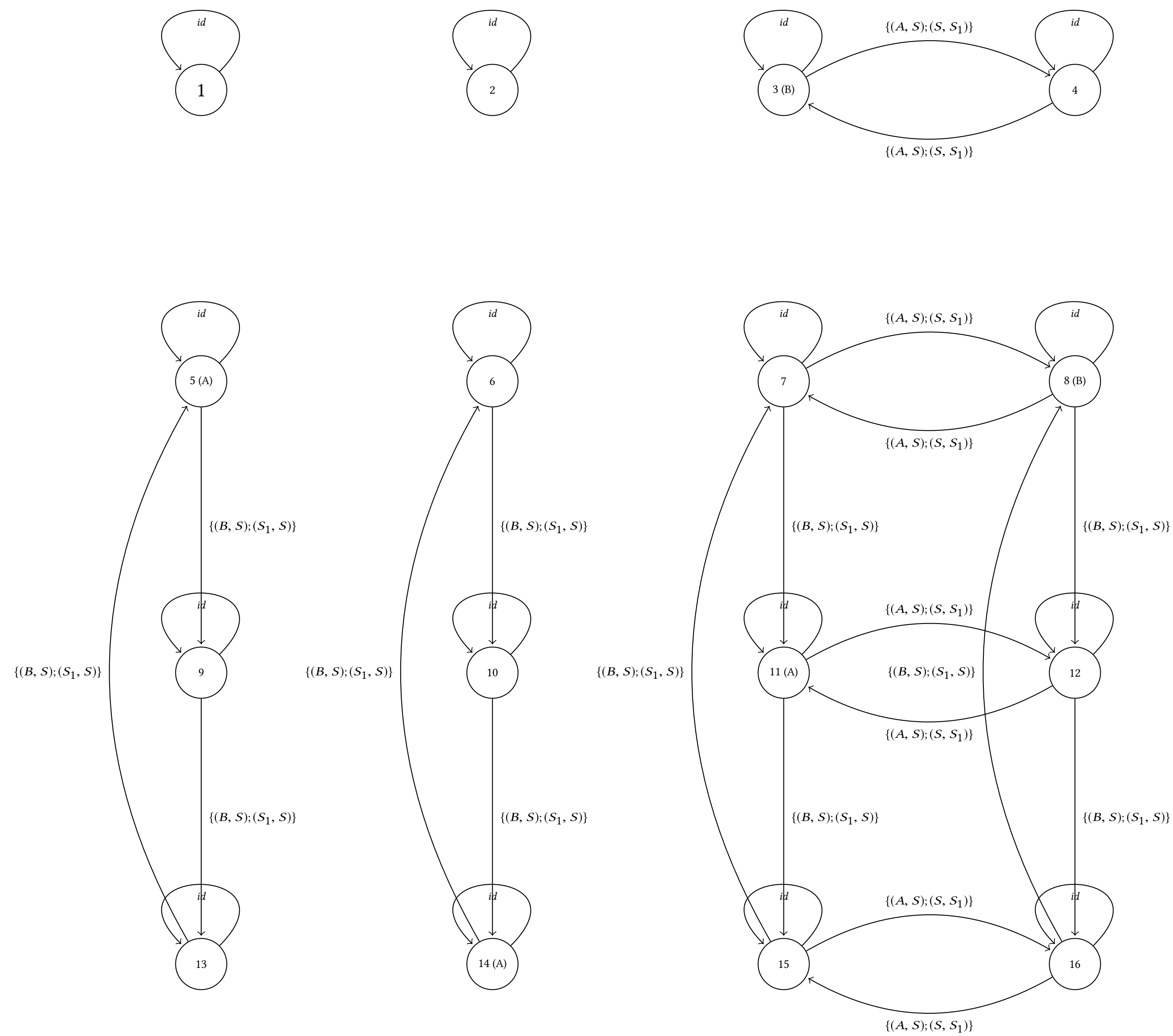


Figure 7: Grid with self-loops

- Our goal is find valid nonterminals for each vertex in $\mathcal{G}$. We can do it iteratively: we can check validity of nonterminals in final vertices of all paths from $\mathcal{G}'_H$ (or $\mathcal{G}'_V$) by multiplication on matrix $X : X[i,j] = \{(N_l, N_l) | N_l$ is known to be valid in $\mathcal{G}(i,j)\}$. Formally we can define next block as one step of iteration.
  - $X = X + X * \mathcal{G}'_H$
  - $X = X + X * \mathcal{G}'_V$
  - Update *IMPLIED* relation and $\mathcal{G}$

  This iteration process all paths with at most one new "zig-zag".
- We should repeat previous step until all path of required length not processed.
- As far as our query is a Dyck query we (hope that we) can use the technique from [1] for estimation of iteration numbers. We can not use it "as is" but we can see, that structure of paths in $\mathcal{G}$ is related to "Pyramids and Valleys" structure from [1].

5 ALGEBRAIC VIEW

Steps for reduction of our problem to purely algebraic problem.

(1) Utilize Rytter's [3] ideas to construct a grid graph $\mathcal{G}$. All are similar to the linear input parsing, with some detales.
  (a) We use states numbers instead of positions.
  (b) To do it we should guarantee that state numbers are in $[0..n-1]$.
  (c) As a result, grid graph can has cycles.
  (d) Edges congruation property still holds.

(2) We can see, that $\mathcal{G}$ is a Cartezian product of two graphs: $\mathcal{G}_H$ (a horisontal row of $\mathcal{G}$) and $\mathcal{G}_V$ (a vertical row of $\mathcal{G}$) with respective adjacency matrices. Adjacency matrix of $\mathcal{G}$ is $M(\mathcal{G}) = M(\mathcal{G}_V) \otimes I + I \otimes M(\mathcal{G}_H)$ where I is identity matrix of size $n \times n$ and $\otimes$ is a Kronecker product.

(3) Instead of SSSP in the Rytter's algorithm we should compue APSP as a atomic step. We should to do it beacause there is now start position in the graph (FSA). Then we should proof that the number of such steps is $O(\log n)$. Thus we want to compute $\text{vec}(X) * M(\mathcal{G})^k = \text{vec}(X) * [M(\mathcal{G}_V) \otimes I + I \otimes M(\mathcal{G}_H)]^k$. Where X is a matrix of already proved facts, and $M(\mathcal{G})^k$ is a transitive closure of the adjacency matrix of the $\mathcal{G}$. Is it possible to do it in $\widetilde{O}(BMM(n))$?

(4) Note that instead of $(B^T \otimes A) * \text{vec}(X) = \text{vec}(C)$ we can solve $A * X * B = C$ (one of fundamental properties of equitations with Kronecker product [4]). The idea is to use this property. In our case it helps to reduce multiplication of $n^2 \times n^2$ matrices to multiplication of $n \times n$ matrices. **But** multiplication in our semiring is noncommutative. Namely, weights are from noncommutative idempotatnt semiring. So we need to investigate properties of Kronecker product over such semiring. Related research by Thomas Reps: "Newtonian Program Analysis via Tensor Product" [**?**]

6 TWO BRS

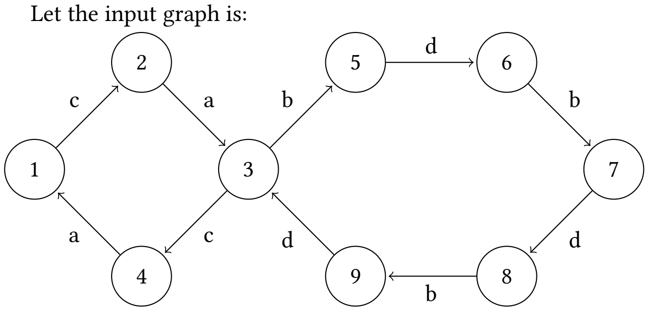
Let the input grammar is

$$S \rightarrow a\ S\ b$$
$$S \rightarrow c\ S\ d$$
$$S \rightarrow a\ b$$
$$S \rightarrow c\ d$$

The input grammar in CNF is

$$S \rightarrow A\ S_1$$
$$S_1 \rightarrow S\ B$$
$$S \rightarrow C\ S_2$$
$$S_2 \rightarrow S\ D$$
$$S \rightarrow C\ D$$
$$S \rightarrow A\ B$$
$$C \rightarrow c$$
$$D \rightarrow d$$
$$A \rightarrow a$$
$$B \rightarrow b$$

Let the input graph is:



REFERENCES

[1] Phillip G Bradford. 2018. Efficient exact paths for Dyck and semi-Dyck labeled path reachability. *arXiv preprint arXiv:1802.05239* (2018).
[2] Krishnendu Chatterjee, Bhavya Choudhary, and Andreas Pavlogiannis. 2017. Optimal Dyck Reachability for Data-dependence and Alias Analysis. *Proc. ACM Program. Lang.* 2, POPL, Article 30 (Dec. 2017), 30 pages. https://doi.org/10.1145/3158118
[3] Wojciech Rytter. 1995. Context-free recognition via shortest paths computation: a version of Valiant's algorithm. *Theoretical Computer Science* 143, 2 (1995), 343–352.
[4] Kathrin Schacke. 2004. On the kronecker product. *Master's thesis, University of Waterloo* (2004).