

Rytter-style Algorithm for Context-Free 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

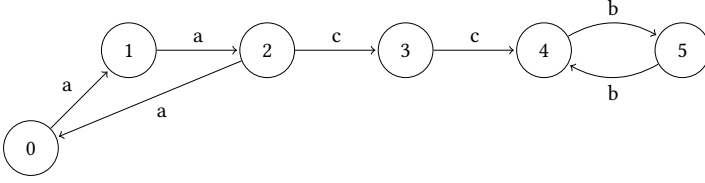


Figure 1: The input graph

ACM Reference Format:

Semyon Grigorev and Ekaterina Shemetova. 2018. Rytter-style Algorithm for Context-Free Path Querying. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

We provide an idea of two steps reduction of CFPQs to Boolean matrix multiplication. First step is reduction of arbitrary CFPQ to Dyck query. Second step is adaptation Rytter's results from [3] for graph. We hope that such reduction helps to get algorithm for CFPQ with $\tilde{O}(BMM(n))$ time complexity where \tilde{O} means polylog factors.

Additionally we discuss “fully algebraic” view on CFPQ complexity which requires investigation of noncommutative structures and matrix spaces over them.

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 $\text{CFPQ}(\mathcal{G}', G')$ to $\text{CFPQ}(\mathcal{G}, G)$. 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. ??.

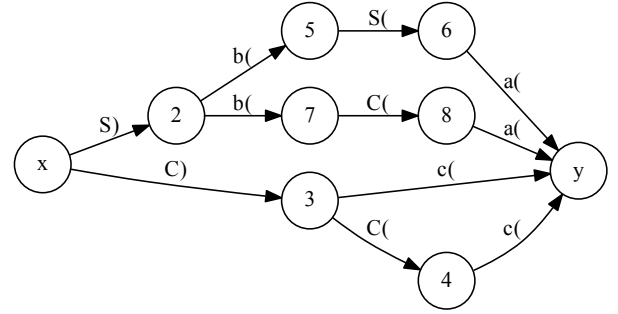


Figure 2: The $M_{\mathcal{G}}$ graph

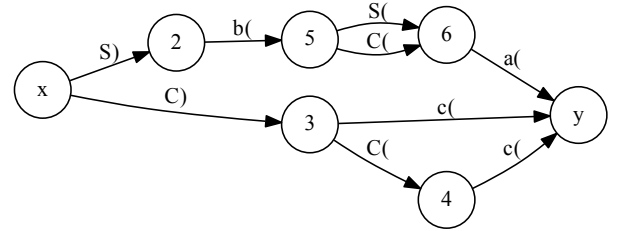


Figure 3: The minimized $M_{\mathcal{G}}$

- (1) Let $\Sigma_0 = \{t_i, t_f \mid t \in \Sigma\}$.
 - (2) Let $N_0 = \{N_i, N_f \mid N \in N\}$.
 - (3) Let $M_{\mathcal{G}} = (V_{\mathcal{G}}, E_{\mathcal{G}}, L_{\mathcal{G}})$ is a directed labeled graph, where $L_{\mathcal{G}} \subseteq (\Sigma_0 \cup N_0)$. 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 $M_{\mathcal{G}}^v$: 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_i, v) for each $v \in V$.
 - And the respective $M_{\mathcal{G}}^v$ for each $v \in V$:
 - reattach all edges outgoing from x^v (“start” vertex of $M_{\mathcal{G}}^v$) to v ;
 - reattach all edges incoming to y^v (“final” vertex of $M_{\mathcal{G}}^v$) to v .
- New input graph is ready. It is presented in fig. 4.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

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

© 2018 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

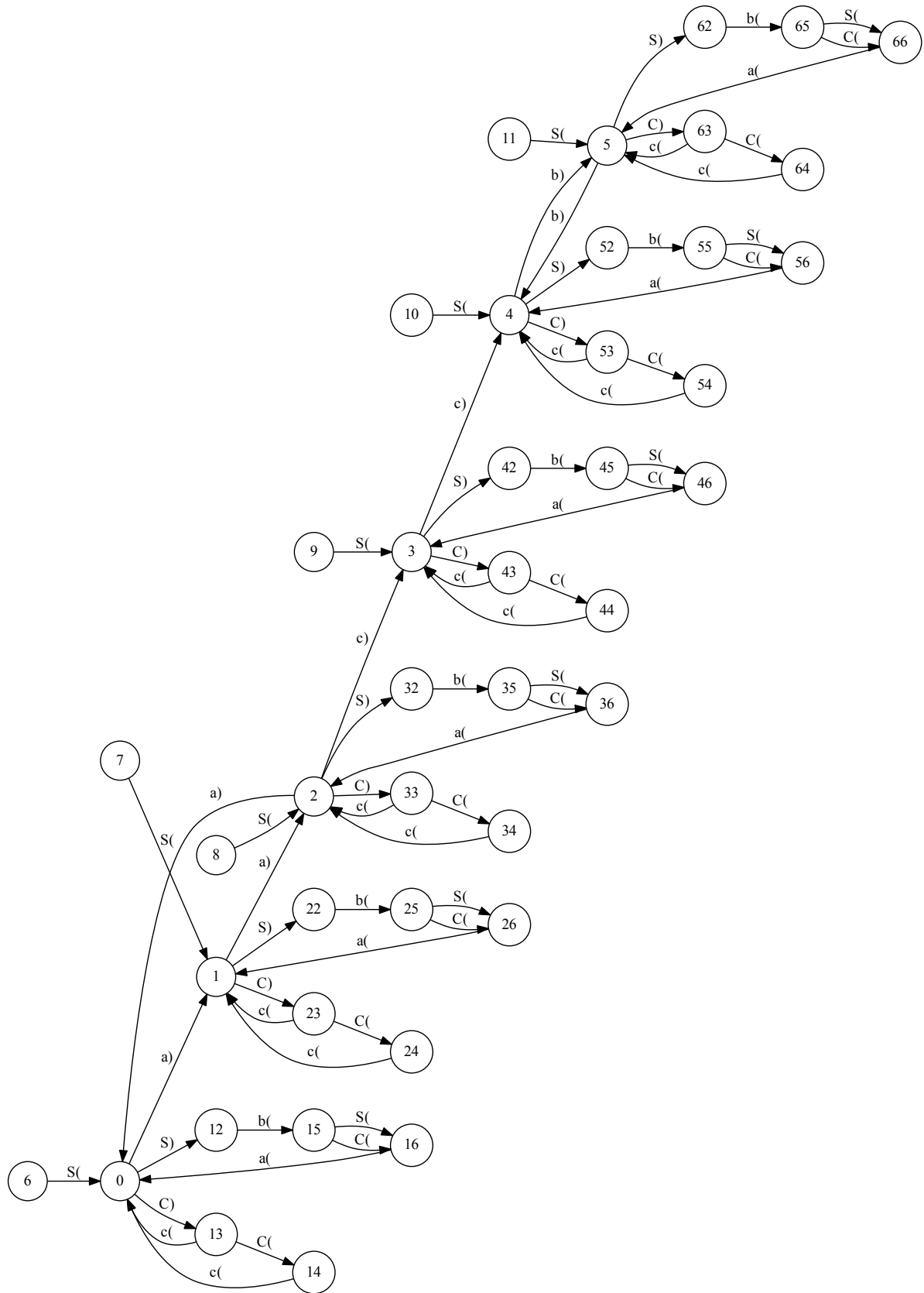


Figure 4: New input graph

- (6) New grammar $\mathcal{G}' = (\Sigma', N', P', S')$ where $\Sigma' = \Sigma_0 \cup N_0$, $N' = \{S'\}$, $P' = \{S' \rightarrow b_l S' b_r; S' \rightarrow b_l b_r \mid b_l, b_r \in \Sigma'\} \cup \{S' \rightarrow S' S'\}$ is a set of productions, $S' \in N'$ is a start nonterminal.

Now, if $\text{CFPQ}(\mathcal{G}', G')$ contains a pair (u'_0, v') such that $e = (u'_0, S_l, u'_1) \in E'$ is an extension edge (step 5, first subitem), then $(u'_1, v') \in \text{CFPQ}(\mathcal{G}, G)$.

In our example, we can find the following path: $7 \xrightarrow{S_l} 1 \xrightarrow{S_r} 22 \xrightarrow{b_l} 25 \xrightarrow{C_l} 26 \xrightarrow{a_l} 1 \xrightarrow{a_r} 2 \xrightarrow{C_l} 33 \xrightarrow{C_r} 34 \xrightarrow{c_l} 2 \xrightarrow{c_r} 3 \xrightarrow{C_l} 43 \xrightarrow{c_r} 3 \xrightarrow{c_l} 4 \xrightarrow{b_r} 5$. Edge $7 \xrightarrow{S_l} 1$ is the extension, so $(1,5)$ should be in $\text{CFPQ}(\mathcal{G}, G)$ and it is true.

3 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.

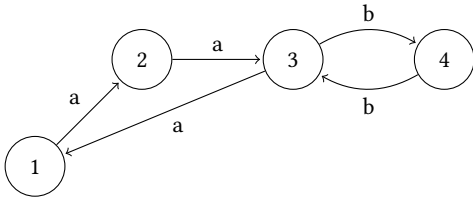
Let the input grammar is

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

The input grammar in CNF is

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

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. Further 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 be 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: $id = \{(N_i, N_i) \mid N_i \in N\}$.

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 $\tilde{O}(BMM(n))$ by using standard techniques for 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 .
- 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 $X : X[i, j] = \{(N_l, N_l) \mid N_l \text{ is known to be valid in } \mathcal{G}(i, j)\}$. 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].

4 ALGEBRAIC VIEW

Steps for reduction of our problem to purely algebraic problem.

- (1) Note that our graph is a Cartesian product of two graph \mathcal{G}_H and \mathcal{G}_V with respective adjacency matrices.
- (2) 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) We want to compute $\text{vec}(X) * M(\mathcal{G})^k = \text{vec}(X) * [M(\mathcal{G}_V) \otimes I + I \otimes M(\mathcal{G}_H)]^k$. Is it possible to do it in $\tilde{O}(BMM(n))$?
- (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 equations 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. So we need to investigate properties of Kronecker product over such semiring. I hope that there are relative results in algebra.

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).

$(B, 2, 3) \Rightarrow (S, 1, 3)$	$(B, 2, 4) \Rightarrow (S, 1, 4)$	$(B, 2, 2) \Rightarrow (S, 1, 2)$	$(B, 2, 1) \Rightarrow (S, 1, 1)$
$(B, 3, 4) \Rightarrow (S, 2, 4)$	$(B, 3, 3) \Rightarrow (S, 2, 3)$	$(B, 3, 2) \Rightarrow (S, 2, 2)$	$(B, 3, 1) \Rightarrow (S, 2, 1)$
$(B, 1, 2) \Rightarrow (S, 3, 2)$	$(B, 1, 3) \Rightarrow (S, 3, 3)$	$(B, 1, 4) \Rightarrow (S, 3, 4)$	$(B, 1, 1) \Rightarrow (S, 3, 1)$
$(S_1, 2, 3) \Rightarrow (S, 1, 3)$	$(S_1, 2, 4) \Rightarrow (S, 1, 4)$	$(S_1, 2, 2) \Rightarrow (S, 1, 2)$	$(S_1, 2, 1) \Rightarrow (S, 1, 1)$
$(S_1, 3, 4) \Rightarrow (S, 2, 4)$	$(S_1, 3, 3) \Rightarrow (S, 2, 3)$	$(S_1, 3, 2) \Rightarrow (S, 2, 2)$	$(S_1, 3, 1) \Rightarrow (S, 2, 1)$
$(S_1, 1, 2) \Rightarrow (S, 3, 2)$	$(S_1, 1, 3) \Rightarrow (S, 3, 3)$	$(S_1, 1, 4) \Rightarrow (S, 3, 4)$	$(S_1, 1, 1) \Rightarrow (S, 3, 1)$
$(A, 2, 3) \Rightarrow (S, 2, 4)$	$(A, 1, 3) \Rightarrow (S, 1, 4)$	$(A, 3, 3) \Rightarrow (S, 3, 4)$	$(A, 4, 3) \Rightarrow (S, 4, 4)$
$(A, 3, 4) \Rightarrow (S, 3, 3)$	$(A, 4, 4) \Rightarrow (S, 4, 3)$	$(A, 2, 4) \Rightarrow (S, 2, 3)$	$(A, 1, 4) \Rightarrow (S, 1, 3)$
$(S, 2, 3) \Rightarrow (S_1, 2, 4)$	$(S, 1, 3) \Rightarrow (S_1, 1, 4)$	$(S, 3, 3) \Rightarrow (S_1, 3, 4)$	$(S, 4, 3) \Rightarrow (S_1, 4, 4)$
$(S, 3, 4) \Rightarrow (S_1, 3, 3)$	$(S, 4, 4) \Rightarrow (S_1, 4, 3)$	$(S, 2, 4) \Rightarrow (S_1, 2, 3)$	$(S, 1, 4) \Rightarrow (S_1, 1, 3)$

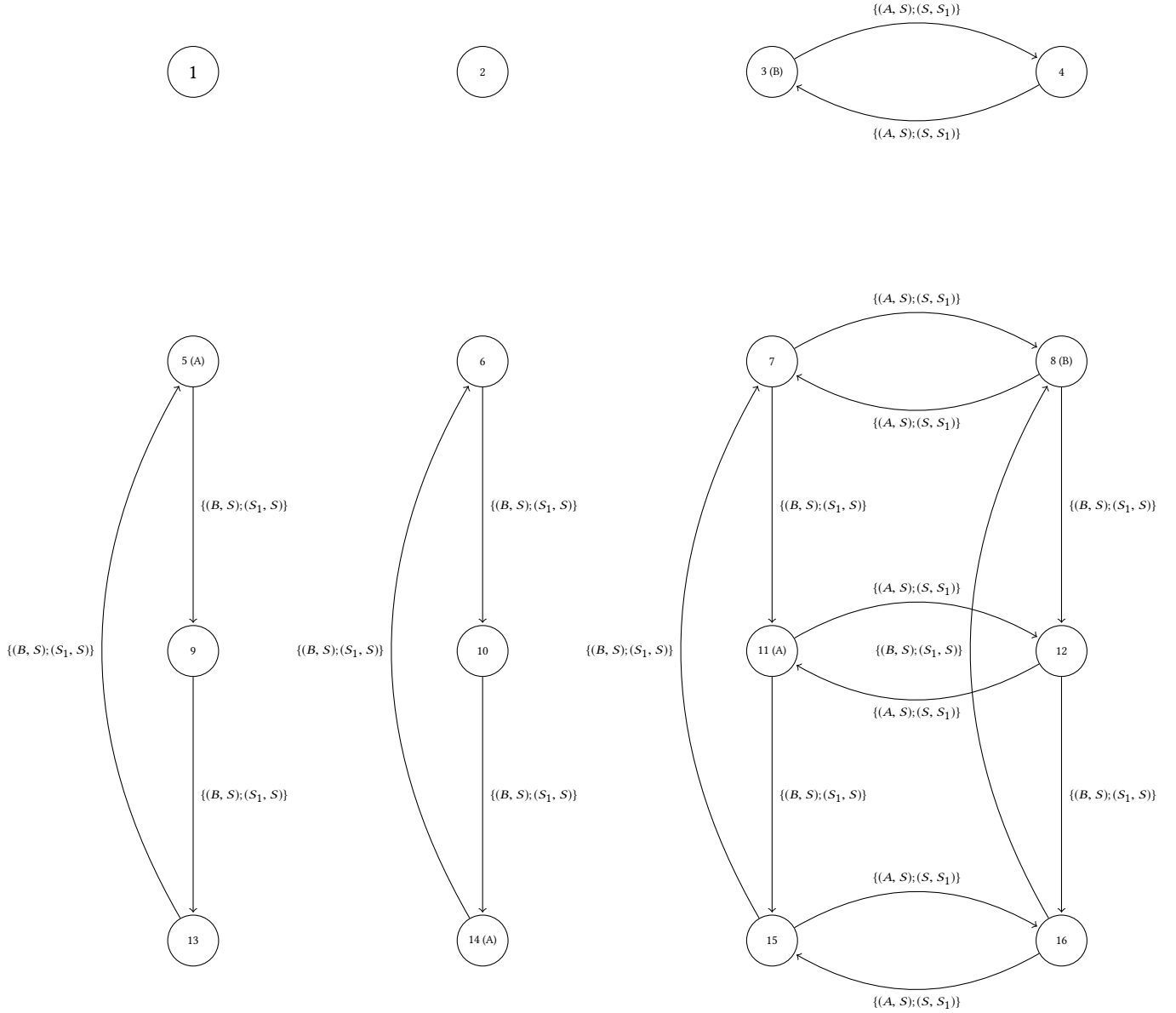
Figure 5: *IMPLIED* relation for our example

Figure 6: Initial grid graph

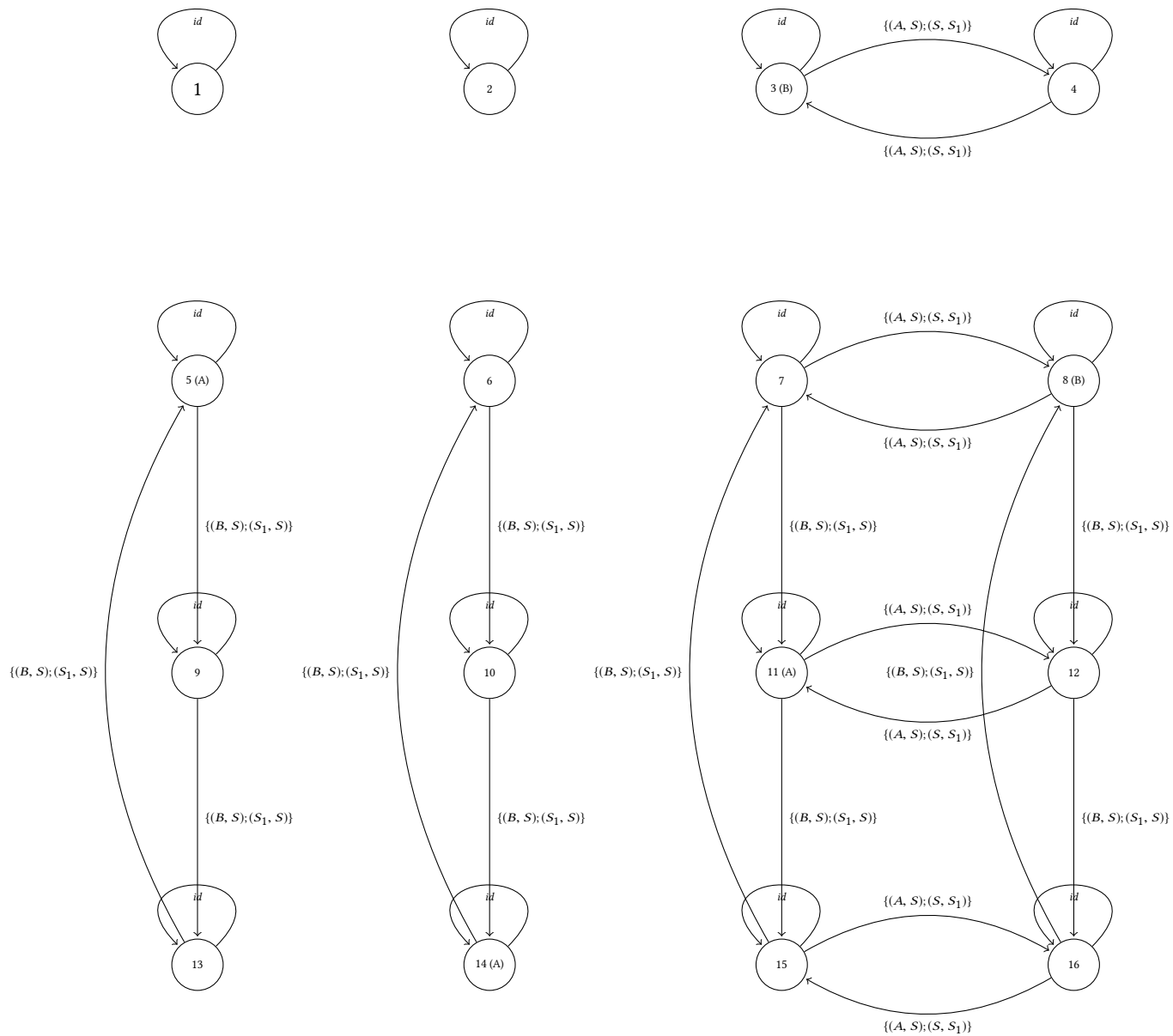


Figure 7: Grid with self-loops