

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

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, ?? pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

The plan.

- Reduction of arbitrary CFPQ to Dyck query.
- Strongly-connected components handling
 - From all pairs reachability to single source reachability
 - Rytter for graph
- Full graph processing

We provide an idea of two steps reduction of CFPQs to Boolean matrix multiplication. First step is reduction of arbitrary CFPQ to Dyck query. The next step is strongly connected components handling. This step is based on Second step is adaptation Rytter's results from [?] 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 [?].

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

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

The input graph is presented in fig. ??.

- (1) Let $\Sigma_0 = \{t_i \mid t_i \in \Sigma\}$.
- (2) Let $N_0 = \{N_i \mid N_i \in N\}$.

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>

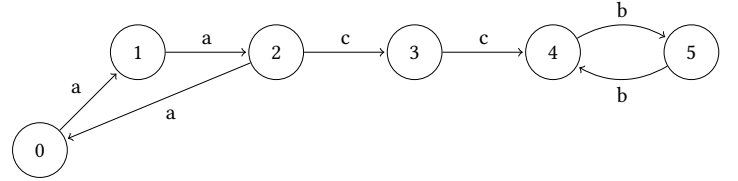


Figure 1: The input graph

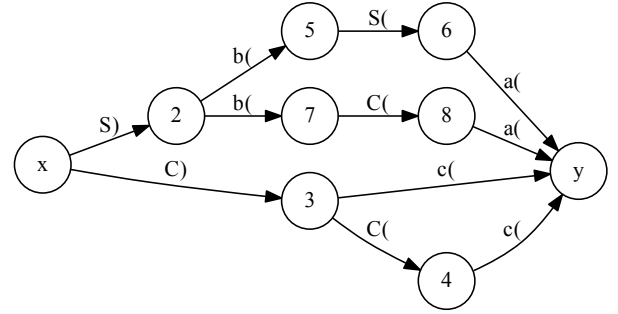


Figure 2: The M_G graph

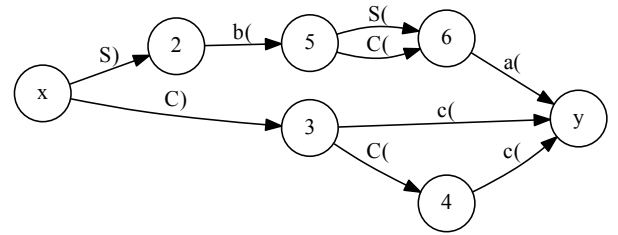


Figure 3: The minimized M_G

- (3) Let $M_G = (V_G, E_G, L_G)$ is a directed labeled graph, where $L_G \subseteq (\Sigma_0 \cup N_0)$. This graph is created the same manner as described in [?] but we do not require the grammar be in CNF. Let $x \in V_G$ and $y \in V_G$ 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_G for our example is presented in fig. ??.

The minimized graph is presented in fig. ??.

- (4) For each $v \in V$ create M_G^v : unique instance of M_G .

- (5) New graph G' is a graph G where each label t is replaced with t_i^j 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. ??.

- (6) New grammar $\mathcal{G}' = (\Sigma', N', P', S')$ where $\Sigma' = \Sigma_0 \cup N_0$, $N' = \{S'\}$, $P' = \{S' \rightarrow b_i S' b_j; S' \rightarrow b_i b_j \mid b_i, b_j \in \Sigma'\} \cup \{S' \rightarrow S' S'\}$ 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 $e = (u'_0, S_i, u'_1) \in E'$ 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: $7 \xrightarrow{S_i} 1 \xrightarrow{S_j} 22 \xrightarrow{b_i} 25 \xrightarrow{C_i} 26 \xrightarrow{a_i} 1 \xrightarrow{a_j} 2 \xrightarrow{C_i} 33 \xrightarrow{C_i} 34 \xrightarrow{c_i} 2 \xrightarrow{c_j} 3 \xrightarrow{C_i} 43 \xrightarrow{c_i} 3 \xrightarrow{c_j} 4 \xrightarrow{b_j} 5$. Edge $7 \xrightarrow{S_i} 1$ is the extension, so $(1,5)$ should be in $\text{CFPQ}(\mathcal{G}, G)$ and it is true.

3 STRONGLY CONNECTED COMPONENTS HANDLING

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

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

Arbitrary path consists of 2-Dyck subpaths connected by unbalanced parts. We can specify grammar for these paths.

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

Now we can start scc processing from one arbitrary selected vertex.

4 RYTTER ALGORITHM FOR GRAPH INPUT

Main idea is to adopt algorithm from [?] 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.

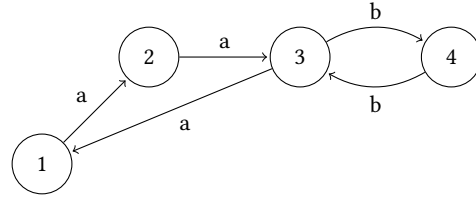
Let the input grammar is

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

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 [?]. The *IMPLIED* relation for our example is presented in figure ??. 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 ??. It can be constructed by the similar way as presented in [?] 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. ??.

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.

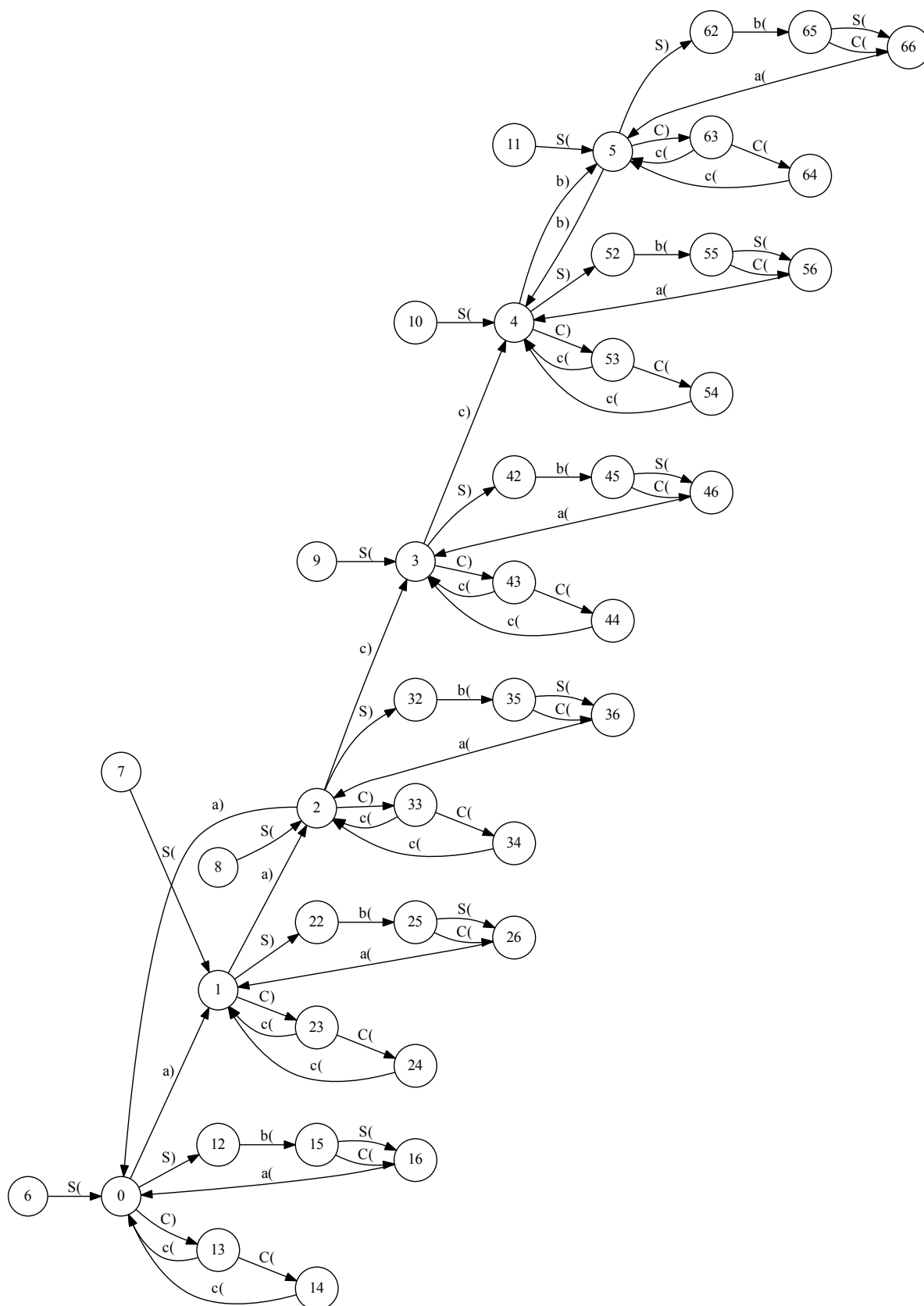


Figure 4: New input graph

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

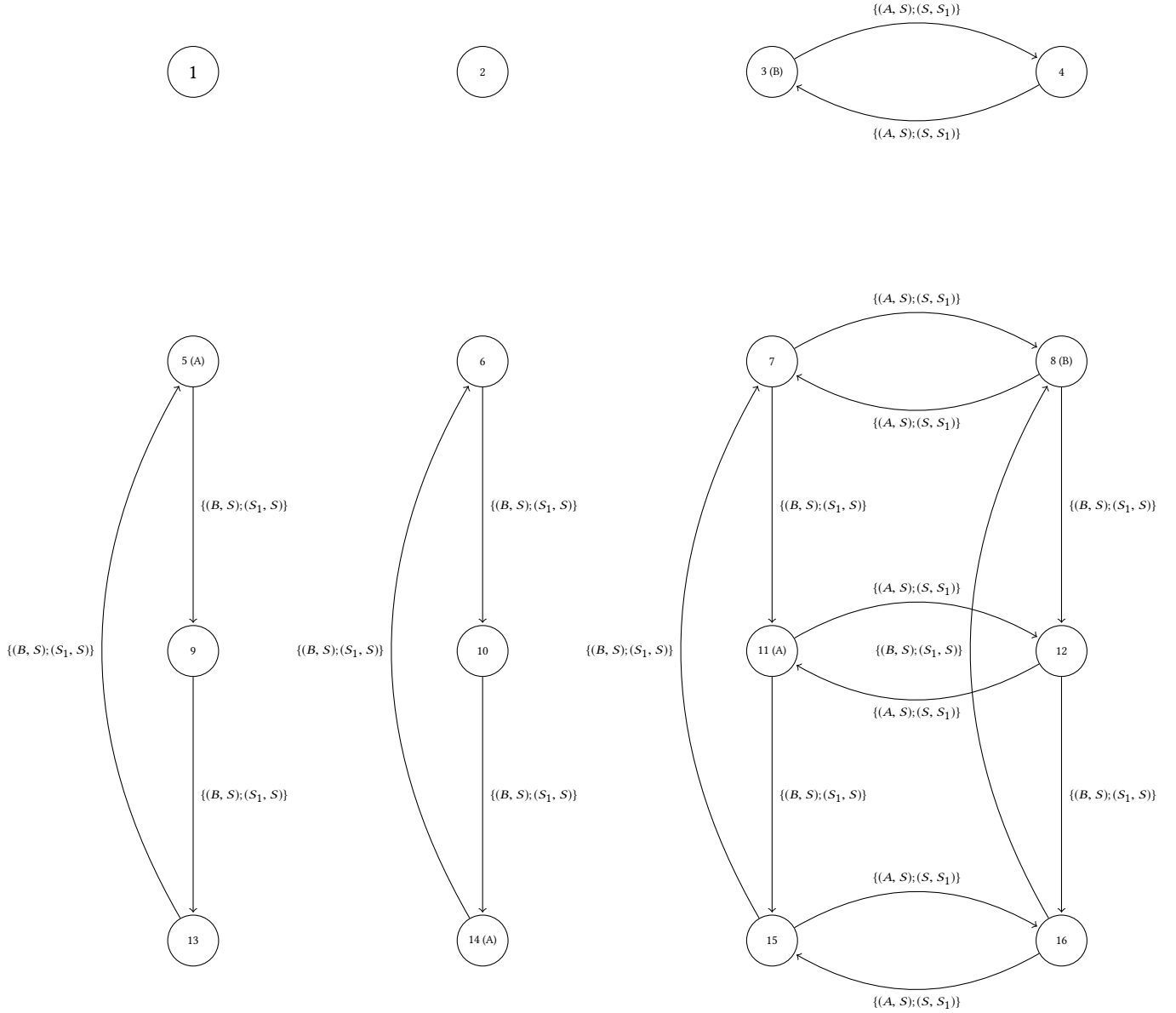
Figure 5: *IMPLIED* relation for our example

Figure 6: Initial grid graph

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:

