Context-Free Path Querying by Using Kronecker Product*

First Author[1[0000−1111−2222−3333]], Second Author[2,3[1111−2222−3333−4444]], and Third Author[3[2222−−3333−4444−5555]]

[1] Princeton University, Princeton NJ 08544, USA
[2] Springer Heidelberg, Tiergartenstr. 17, 69121 Heidelberg, Germany
lncs@springer.com
http://www.springer.com/gp/computer-science/lncs
[3] ABC Institute, Rupert-Karls-University Heidelberg, Heidelberg, Germany
{abc,lncs}@uni-heidelberg.de

Abstract. Abstact is very abstract.

Keywords: Path querying · Graph database · Context-free grammars · CFPQ · Kronecker product · !!! .

1 Introduction

CFPQ is popular and widely used for !!!.

Matrices [2] — algorithm is fast, but grammar size is problem.

Moreover, bad for regualr queryes.

In adition, Kuipers says that existing algorithms are not applicable for real-world problems. So we should develop new ones.

Following contribution.

1. New algorithm is proposed. Recursive automata intersection. Applicable for regular queryes. Linear algebra.
2. Example is provided.
3. Evaluation is prvided.

* Supported by organization x.

2 Recursive State Machines

In this section, we introduce the notation of recursive state machine (or RSM), with its definition and semantic description. This kind of computational machines extends the definition of finite state machines and increases the computational capabilities of this formalism.

From conceptual point of view, RSM behaves as set of finite state machines (or FSM), so called *boxes* or *component state machines* [1], which are executed in classical definition of FSM with additional *recursive calls* and implicit *call stack*, what allows to *call* some component state machine f_2 from f_1, and then return execution flow from f_2 to f_1.

Formally, a recursive state machine R over a finite alphabet Σ is defined as tuple of elements $(M, m, \{C_i\}_{i \in M})$, where:

- M is a finite set of boxes' labels
- m is an initial box label
- Set of *component state machines* or *boxes*, where $C_i = (\Sigma \cup M, Q_i, q_i^0, F_i, \delta_i)$:
  - $\Sigma \cup M$ is set of symbols, $\Sigma \cap M = \emptyset$
  - Q_i is finite set of states, where $Q_i \cap Q_j = \emptyset, \forall i \neq j$
  - q_i^0 is an initial state for component state machine C_i
  - F_i is set of final states for C_i, where $F_i \subseteq Q_i$
  - δ_i is transition function for C_i, where $\delta_i : Q_i \times (\Sigma \cup M) \to Q_i$

Semantic of the execution of such automata R involves a pair of objects (q, S), where S is global stack of *return states* from $\bigcup_{i \in M} Q_i$ such as $S = \langle ...q_r \rangle$, and q is some current state of the machine, where $q \in \bigcup_{i \in M} Q_i$. The execution process starts from box m initial state q_m^0 and empty stack as follows $(q_m^0, \langle\rangle)$. Transitions for a given global machine state (q, S) to some new state (q', S') are defined as follows:

- $q' := q_b', S' := S$, where $\delta_b(q_b, s) \to q_b', s \in \Sigma, q = q_b, S = \langle ...q_r \rangle$
- $q' := q_t^0, S' := \langle ...q_r, q_b' \rangle$, where $\delta_b(q_b, t) \to q_b', t \in M, q = q_b, S = \langle ...q_r \rangle$
- $q' := q_b', S' := \langle ...q_r \rangle$, where $q = q_t^i, q_t^i \in F_t, S = \langle ...q_r, q_b' \rangle$

Accordingly to [1], recursive state machines are equivalent in the general computational capacity to pushdown systems. Since pushdown systems are capable of accepting context-free languages [4], it is clear to use a recursive state machine to encode some grammar G.

As an example of such machine, consider the following recursive state machine R, which is depicted in Figure 1.

For the sake of simplicity, we escape detailed discussion of the conversion of a context free grammar G to a recursive machine R. In the basic case, an algorithm for constructing such recursive state machine could be composed from several stages, where each stage involves building of finite state machine for each non-terminal N from grammar G.

Since R is composed from set of FSM, it could be useful for computational tasks to represent such R as a adjacency matrix, where vertices are states from

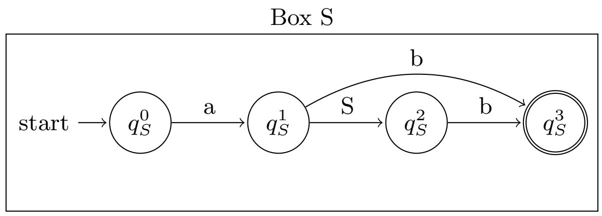

Fig. 1: The recursive state machine R for grammar G

$\bigcup_{i \in M} Q_i$ and edges are transitions between q_i^a and q_i^b with label $l \in \Sigma \cup M$, if $\delta_i(q_i^a, l) = q_i^b$.

An example of such adjacency matrix M_R for our machine R is depicted in Figure 2.

$$M_R = \begin{pmatrix} . & . & \{a\} & . \\ . & . & \{S\} & \{b\} \\ . & . & . & \{b\} \\ . & . & . & . \end{pmatrix}$$

Fig. 2: The adjacency matrix M_R for recursive state machine R

3 Kronecker Product Based CFPQ Algorithm

The idea of the algorithm is based on generalisation of the finite-state machine intersection for a recursive automata, created from input grammar, and an input graph. The result of the intersection is evaluated as a Kronecker product of the corresponding adjacency matrices for automata and graph. To solve reachability problem it is enough to represent intersection result as a Boolean matrix, what simplifies algorithm implementation and allows to express it in terms of basic matrix operations. Listing 1. shows main steps of the solution.

As an input algorithm accepts context-free grammar $G = (\Sigma, N, P)$ and graph $\mathcal{G} = (V, E, L)$. All sets in the grammar and graph are supposed to be of finite size. Recursive automata R is created from G. The process of the creation is out of the scope of this article. M_1 and M_2 are the adjacency matrices for automata R and graph $\mathcal{G}$ correspondingly. Cell values of this matrices could be represented as sets of elements from $L \cup N \cup \Sigma$.

Then for each vertex i of the graph $\mathcal{G}$ the algorithm adds loops with non-terminals, which allows to derive ε-word. Here the rule is implied: each vertex of the graph is reachable by itself through ε-transition. Since the automata R does not have ε-transitions, the ε-word could be derived only if some state

s of the R is initial and final for some non-terminal. This info is queried by $getNonterminals()$ function for each state s correspondingly.

The algorithm is executed while matrix M_2 is changing. For each iteration Kronecker product of matrices M_1 and M_2 is evaluated. The result is saved in M_3 as a Boolean matrix. For given M_3 evaluated C_3 matrix via $transitiveClosure()$ function call. The M_3 could be interpreted as an adjacency matrix for an oriented graph without labels, used to evaluate transitive closure in terms of classical graph definition of this operation. Then the algorithm iterates over cells of the C_3. For pair of indices (i, j) computes s and f - initial and final states in recursive automata R which relate to the concrete $C_3[i, j]$ of the tensor matrix. Then algorithm checks whether for given s and f states automata has at least one non-terminal path. If the conditional statement is *true* then algorithm adds non-terminals of that path via $getNonterminals()$ to the concrete cell of the adjacency matrix M_2 of the graph.

As a result the algorithm returns updated matrix M_2 which contains initial graph $\mathcal{G}$ data and non-terminals from N. If a cell $M_2[i, j]$ for any valid indices i and j contains symbol $S \in N$, therefore, vertex j is reachable from vertex i in grammar G for non-terminal S.

Listing 1 Kronecker product based CFPQ

```
function contextFreePathQuerying(G, 𝒢)
    R ← Recursive automata for G
    M_1 ← Adjacency matrix for R
    M_2 ← Adjacency matrix for 𝒢
    for s ∈ 0..dim(M_1) − 1 do
        for i ∈ 0..dim(M_2) − 1 do
            M_2[i, i] ← M[i, i]_2 ∪ getNonterminals(R, s, s)
    while Matrix M_2 is changing do
        M_3 ← M_1 ⊗ M_2                              ▷ Evaluate tensor product
        C_3 ← transitiveClosure(M_3)
        n ← dim(M_3)                                 ▷ Matrix M_3 size = n × n
        for i ∈ 0..n − 1 do
            for j ∈ 0..n − 1 do
                if C_3[i, j] then
                    s, f ← getStates(C_3, i, j)
                    if getNonterminals(R, s, f) ≠ ∅ then
                        x, y ← getCoordinates(C_3, i, j)
                        M_2[x, y] ← M_2[x, y] ∪ getNonterminals(R, s, f)
    return M_2
```

Since the Kronecker product evaluated in the fixed order, such as $M_1 \otimes M_2$, the functions $getStates$ and $getCoordinates$ could be implemented as shown in Listing 2. This implementation appeals to the blocked structure of the matrix C_3, where each block corresponds to some automata and graph edge.

Listing 2 Help functions for Kronecker product based CFPQ

```
function GETSTATES(C, i, j)
    r ← dim(M_1)                          ▷ M_1 is adjacency matrix for automata R
    return ⌊i/r⌋, ⌊j/r⌋
function GETCOORDINATES(C, i, j)
    n ← dim(M_2)                          ▷ M_2 is adjacency matrix for graph 𝒢
    return i mod n, j mod n
```

3.1 Example

This section is intended to provide step-by-step demonstration of the proposed algorithm. As an example consider the following query, theoretical worst case for CFPQ time complexity, proposed by J.Hellings [3]: graph $\mathcal{G}$ presented in Figure 3 and context-free grammar $G = (\Sigma, N, P)$ for a language $\{a^n b^n \mid n \geq 1\}$ where:

- Set of terminals $\Sigma = \{a, b\}$.
- Set of non-terminals $V = \{S\}$.
- Set of production rules $P = \{S \to aSb, S \to ab\}$.

Since the proposed algorithm processes grammar in form of recursive automata, we first provide automata R in Figure 1. The initial state of the automata is (0), the final state is (3). The notation $\{S\}$ denotes here that non-terminal S could be derived in automata path from vertex (0) to (3).

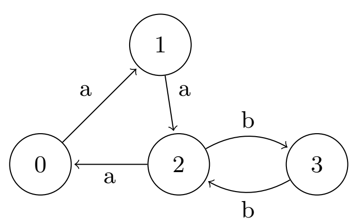

Fig. 3: The input graph $\mathcal{G}$ for example query

Adjacency matrices M_1 and M_2 for automata R and graph $\mathcal{G}$ respectively are initialised as follows:

$$M_1 = \begin{pmatrix} . & . & \{a\} & . \\ . & . & \{S\} & \{b\} \\ . & . & . & \{b\} \\ . & . & . & . \end{pmatrix}, \qquad M_2^0 = \begin{pmatrix} . & \{a\} & . & . \\ . & . & \{a\} & . \\ \{a\} & . & . & \{b\} \\ . & . & \{b\} & . \end{pmatrix}.$$

After all the data is initialised in lines **2–4**, the algorithm handles ε-case. Because automata R does not have ε-transitions and ε-word is not included in grammar G language lines **5–7** of the algorithm do no affect the input data.

Then the algorithm enters while loop and iterates as long as matrix M_2 is changing. We provide step-by-step evaluation of matrices M_3, C_3 and updating of matrix M_2. All the matrices are denoted with upper index of the current loop iteration. The first loop iteration is indexed as 1.

For the first while loop iteration the tensor product $M_3^1 = M_1 \otimes M_2^0$ and transitive closure C_3^1 are evaluated as follows:

$$M_3^1 = \left(\begin{array}{cccc|cccc|cccc|cccc}
\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\hline
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot\\
\hline
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot\\
\hline
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot
\end{array}\right), \quad
C_3^1 = \left(\begin{array}{cccc|cccc|cccc|cccc}
\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\mathbf{1}\\
\cdot&\cdot&\cdot&\cdot&1&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\hline
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot\\
\hline
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&1&\cdot\\
\hline
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot\\
\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot&\cdot
\end{array}\right).$$

Note here that the dimension n of the matrix M_3 equals 16, and this value is constant in time of the algorithm execution.

After the transitive closure evaluation matrix C_3^1 cell $(1, 15)$ contains non-zero value. It means that vertex with index 15 is accessible from vertex with index 1 in a graph, represented by adjacency matrix M_3^1.

Then the algorithm lines **14**–**18** are executed. In that section algorithm adds non-terminals to the graph matrix M_2^1. Because this step is additive we are only interested in newly appeared values in matrix C_3^1 such as value $C_3^1[1, 15]$.

For the value $C_3^1[1, 15]$:

- Indices of the automata vertices $s = 0$ and $f = 3$, because value $C_3^1[1, 15]$ located in upper right matrix block $(0, 3)$.
- Indices of the graph vertices $x = 1$ and $y = 3$ are evaluated as value $C_3^1[1, 15]$ indices relatively to its block $(0, 3)$.
- Function call $hasPathForNonterminals()$ returns **true** since the automata R has path for non-terminal S from vertex 0 to 3.
- Function call $getNonterminals()$ returns $\{S\}$ since this is the only non-terminal which could be derived in path from vertex 0 to 3.

After the first loop iteration matrix symbol S is added to the cell $M_2^1[1, 3]$. It is relevant data, because initial graph has path $1 \to 2 \to 3$ which could be derived for S. The updated matrix and graph are depicted in Figure 4.

For the second loop iteration matrices M_3^2 and C_3^2 are evaluated as listed in Figure 5. For this iteration in the matrix C_3^2 appeared new non-zero values in cells with indices $[0, 11]$, $[0, 14]$ and $[5, 14]$. Because only the cell value with index $[0, 14]$ corresponds to the automata path with not empty non-terminal set $\{S\}$ its data affects adjacency matrix M_2. The updated matrix and graph $\mathcal{G}$ are depicted in Figure 6.

$$M_2^1 = \begin{pmatrix} . & \{a\} & . & . \\ . & . & \{a\} & \{\mathbf{S}\} \\ \{a\} & . & . & \{b\} \\ . & . & \{b\} & . \end{pmatrix}$$

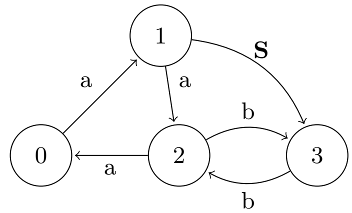

Fig. 4: The updated matrix M_2^1 and graph $\mathcal{G}$ after first loop iteration for example query

$$M_3^2 = \left(\begin{array}{cccc|cccc|cccc|cccc}
. & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & . \\
. & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
\hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & \boxed{\mathbf{1}} & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\
\hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\
\hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & .
\end{array}\right)
\quad
C_3^2 = \left(\begin{array}{cccc|cccc|cccc|cccc}
. & . & . & . & . & 1 & . & . & . & . & . & \boxed{\mathbf{1}} & . & . & \boxed{\mathbf{1}} & . \\
. & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & 1 \\
. & . & . & . & 1 & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
\hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & 1 & . & . & \boxed{\mathbf{1}} & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\
\hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\
\hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & .
\end{array}\right)$$

Fig. 5: The second iteration tensor product and transitive closure evaluation for example query

The remaining matrices C_3 and M_2 for the algorithm main loop execution are listed in the Figure 7 and Figure 8 correspondingly. For the sake of simplicity evaluated matrices M_3 are not included because its computation is a straightforward process. The last loop iteration is 7. Although the matrix M_2^6 is updated with new non-terminal S for the cell $[2, 2]$ after transitive closure evaluation the new values to the matrix M_2 is not added. Therefore matrix M_2 has stopped changing and the algorithm is successfully finished.

For the example query algorithm takes 7 iterations for the *while*-loop. The updated graph $\mathcal{G}$ is depicted in the Figure 9.

4 Evaluation

RedisGraph + CFPQ_Data

Cases, when kronecker should be significantly better that matrix. When grammar is big. When query is regular.

5 Conclusion

We present !!!

$$M_2^2 = \begin{pmatrix} . & \{a\} & \{\mathbf{S}\} & . \\ . & . & \{a\} & \{S\} \\ \{a\} & . & . & \{b\} \\ . & . & \{b\} & . \end{pmatrix}$$

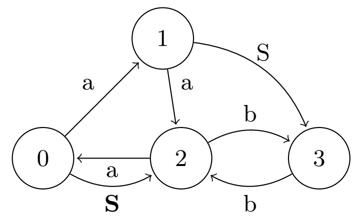

Fig. 6: The updated matrix M_2^2 and graph $\mathcal{G}$ after second loop iteration for example query

$$C_3^3 = \left(\begin{array}{cccc|cccc|cccc|cccc}
. & . & . & . & . & 1 & . & . & . & . & . & 1 & . & . & 1 & . \\
. & . & . & . & . & . & 1 & . & . & . & . & . & . & . & . & 1 \\
. & . & . & . & 1 & . & . & . & . & . & \mathbf{1} & . & . & . & . & \mathbf{1} \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\ \hline
. & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & \mathbf{1} \\
. & . & . & . & . & . & . & . & . & . & . & 1 & . & . & 1 & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\ \hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\ \hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & .
\end{array}\right)
C_3^4 = \left(\begin{array}{cccc|cccc|cccc|cccc}
. & . & . & . & . & 1 & . & . & . & . & . & 1 & . & . & 1 & . \\
. & . & . & . & . & . & 1 & . & . & . & . & \mathbf{1} & . & . & \mathbf{1} & 1 \\
. & . & . & . & 1 & . & . & . & . & . & 1 & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\ \hline
. & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & 1 & . & . & 1 & . \\
. & . & . & . & . & . & . & . & . & . & . & 1 & . & . & \mathbf{1} & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\ \hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\ \hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & .
\end{array}\right)$$

$$C_3^5 = \left(\begin{array}{cccc|cccc|cccc|cccc}
. & . & . & . & . & 1 & . & . & . & . & \mathbf{1} & 1 & . & . & 1 & \mathbf{1} \\
. & . & . & . & . & . & 1 & . & . & . & . & 1 & . & . & 1 & 1 \\
. & . & . & . & 1 & . & . & . & . & . & 1 & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\ \hline
. & . & . & . & . & . & . & . & . & . & 1 & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & 1 & 1 & . & . & 1 & \mathbf{1} \\
. & . & . & . & . & . & . & . & . & . & . & 1 & . & . & 1 & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\ \hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\ \hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & .
\end{array}\right)
C_3^6 = \left(\begin{array}{cccc|cccc|cccc|cccc}
. & . & . & . & . & 1 & . & . & . & . & 1 & 1 & . & . & 1 & 1 \\
. & . & . & . & . & . & 1 & . & . & . & . & 1 & . & . & 1 & 1 \\
. & . & . & . & 1 & . & . & . & . & . & 1 & \mathbf{1} & . & . & \mathbf{1} & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\ \hline
. & . & . & . & . & . & . & . & . & . & 1 & 1 & . & . & \mathbf{1} & 1 \\
. & . & . & . & . & . & . & . & . & . & 1 & 1 & . & . & 1 & 1 \\
. & . & . & . & . & . & . & . & . & . & . & 1 & . & . & 1 & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\ \hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & 1 & . \\ \hline
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & . \\
. & . & . & . & . & . & . & . & . & . & . & . & . & . & . & .
\end{array}\right)$$

Fig. 7: Transitive closure for $3-6$ loop iterations for example query

$$M_2^3 = \begin{pmatrix} . & \{a\} & \{S\} & . \\ . & . & \{a\} & \{S\} \\ \{a\} & . & . & \{b, \mathbf{S}\} \\ . & . & \{b\} & . \end{pmatrix}
M_2^4 = \begin{pmatrix} . & \{a\} & \{S\} & . \\ . & . & \{a, \mathbf{S}\} & \{S\} \\ \{a\} & . & . & \{b, S\} \\ . & . & \{b\} & . \end{pmatrix}$$

$$M_2^5 = \begin{pmatrix} . & \{a\} & \{S\} & \{\mathbf{S}\} \\ . & . & \{a, S\} & \{S\} \\ \{a\} & . & . & \{b, S\} \\ . & . & \{b\} & . \end{pmatrix}
M_2^6 = \begin{pmatrix} . & \{a\} & \{S\} & \{S\} \\ . & . & \{a, S\} & \{S\} \\ \{a\} & . & \{\mathbf{S}\} & \{b, S\} \\ . & . & \{b\} & . \end{pmatrix}$$

Fig. 8: The updated matrix M_2 for $3-6$ loop iterations for example query

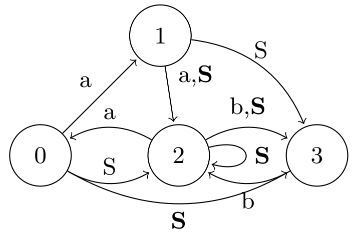

Fig. 9: The result graph $\mathcal{G}$ for example query

Future research. Performance improvements. Detailed investigation of the algorithm formal properies such as time and space complexity. GraphBLAST. Paths, not just reachability.

References

1. Alur, R., Etessami, K., Yannakakis, M.: Analysis of recursive state machines. In: Berry, G., Comon, H., Finkel, A. (eds.) Computer Aided Verification. pp. 207–220. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
2. Azimov, R., Grigorev, S.: Context-free path querying by matrix multiplication. In: Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA). pp. 5:1–5:10. GRADES-NDA '18, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3210259.3210264, http://doi.acm.org/10.1145/3210259.3210264
3. Hellings, J.: Querying for paths in graphs using context-free path queries. arXiv preprint arXiv:1502.02242 (2015)
4. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation (3rd Edition). Addison-Wesley Longman Publishing Co., Inc., USA (2006)