Conjunctive Path Querying by Matrix Multiplication⋆

Rustam Azimov[1] and Semyon Grigorev[1]

Saint Petersburg State University, St. Petersburg, Russia
st013567@student.spbu.ru
s.v.grigoriev@spbu.ru

Abstract. **Keywords:**

1 Introduction

Also, there are conjunctive grammars [8], which have more expressive power than context-free grammars. Path querying with conjunctive grammars is known to be undecidable [5]. Although there is an algorithm [14] for path querying with linear conjunctive grammars [8] which is used in static analysis and provides an over-approximation of the result. However, there is no algorithm for path querying with conjunctive grammars of an arbitrary form.

The purpose of this work is to develop an effective matrix-based algorithm for path querying with conjunctive grammars which allows us to effectively apply GPGPU computing techniques.

2 Preliminaries

In this section, we introduce the basic notions used throughout the paper.

Let Σ be a finite set of edge labels. Define an *edge-labeled directed graph* as a tuple $D = (V, E)$ with a set of nodes V and a directed edge-relation $E \subseteq V \times \Sigma \times V$. For a path π in a graph D, we denote the unique word obtained by concatenating the labels of the edges along the path π as $l(\pi)$. Also, we write $n\pi m$ to indicate that a path π starts at the node $n \in V$ and ends at the node $m \in V$.

We deviate from the usual definition of a conjunctive grammar in the *binary normal form* [8] by not including a special start non-terminal, which will be specified in the queries to the graph. Since every conjunctive grammar can be transformed into an equivalent one in the binary normal form [8] and checking that an empty string is in the language is trivial, then it is sufficient to only consider grammars of the following type. A *conjunctive grammar* is 3-tuple $G = (N, \Sigma, P)$ where N is a finite set of non-terminals, Σ is a finite set of terminals, and P is a finite set of productions of the following forms:

⋆ Supported by the Russian Science Foundation grant 18-11-00100 and a grant from JetBrains Research.

– $A \to B_1C_1 \ \& \ldots \& \ B_mC_m$, for $m \geq 1$, $A, B_i, C_i \in N$,
– $A \to x$, for $A \in N$ and $x \in \Sigma$.

For conjunctive grammars we use the conventional notation $A \xrightarrow{*} w$ to denote that the string $w \in \Sigma^*$ can be derived from a non-terminal A by some sequence of applying the production rules from P. The relation $\to$ is defined as follows:

– Using a rule $A \to B_1C_1 \ \& \ldots \& \ B_mC_m \in P$, any atomic subterm A of any term can be rewritten by the subterm $(B_1C_1 \ \& \ldots \& \ B_mC_m)$:

$$\ldots A \ldots \to \ldots (B_1C_1 \ \& \ldots \& \ B_mC_m) \ldots$$

– A conjunction of several identical strings in Σ^* can be rewritten by one such string: for every $w \in \Sigma^*$,

$$\ldots (w \ \& \ldots \& \ w) \ldots \to \ldots w \ldots$$

The *language* of a conjunctive grammar $G = (N, \Sigma, P)$ with respect to a start non-terminal $S \in N$ is defined by $L(G_S) = \{w \in \Sigma^* \mid S \xrightarrow{*} w\}$.

For a given graph $D = (V, E)$ and a conjunctive grammar $G = (N, \Sigma, P)$, we define *conjunctive relations* $R_A \subseteq V \times V$, for every $A \in N$, such that $R_A = \{(n, m) \mid \exists n\pi m \ (l(\pi) \in L(G_A))\}$. Note that this is similar to the definition of the context-free relations which is used for the context-free path querying [5].

We define a *conjunctive matrix multiplication*, $a \circ b = c$, where a and b are matrices of the suitable size that have subsets of N as elements, as $c_{i,j} = \{A \mid \exists (A \to B_1C_1 \ \& \ldots \& \ B_mC_m) \in P$ such that $(B_k, C_k) \in d_{i,j}\}$, where $d_{i,j} = \bigcup_{k=1}^{n} a_{i,k} \times b_{k,j}$.

Also, we define the *conjunctive transitive closure* of a square matrix a as $a^{conj} = a^{(1)} \cup a^{(2)} \cup \cdots$ where $a^{(i)} = a^{(i-1)} \cup (a^{(i-1)} \circ a^{(i-1)})$, $i \geq 2$ and $a^{(1)} = a$.

3 Related works

Problems in many areas can be reduced to one of the formal-languages-constrained path problems [1]. For example, various problems of static code analysis [2,13] can be formulated in terms of the context-free language reachability [9] or in terms of the linear conjunctive language reachability [14].

One of the well-known problems in the area of graph database analysis is the language-constrained path querying. For example, the regular language constrained path querying [10,**?**,**?**,**?**], and the context-free language constrained path querying.

There are a number of solutions [5,**?**,**?**] for context-free path query evaluation w.r.t. the relational query semantics, which employ such parsing algorithms as CYK [7,**?**] or Earley [4]. Other examples of path query semantics are *single-path* and *all-path query semantics*. The all-path query semantics requires presenting all possible paths from node m to node n whose labeling is derived from a non-terminal A for all triples (A, m, n) evaluated using the relational query semantics. While the single-path query semantics requires presenting only one such a path for all the node-pairs (m, n). Hellings [6] presented algorithms for the context-free path query evaluation using the single-path and the all-path query semantics. If a context-free path query w.r.t. the all-path query semantics is evaluated on cyclic graphs, then the query result can be an infinite set of paths. For this reason, in [6], annotated grammars are proposed as a possible solution.

In [3], the algorithm for context-free path query evaluation w.r.t. the all-path query semantics is proposed. This algorithm is based on the generalized top-down parsing algorithm — GLL [11]. This solution uses derivation trees for the result representation which is more native for grammar-based analysis. The algorithms in [3,**?**] for the context-free path query evaluation w.r.t. the all-path query semantics can also be used for query evaluation using the relational and the single-path semantics.

Our work is inspired by Valiant [12], who proposed an algorithm for general context-free recognition in less than cubic time. This algorithm computes the same parsing table as the CYK algorithm but does this by offloading the most intensive computations into calls to a Boolean matrix multiplication procedure. This approach not only provides an asymptotically more efficient algorithm but it also allows us to effectively apply GPGPU computing techniques. Valiant's algorithm computes the transitive closure of a square upper triangular matrix.

Hellings [5] presented an algorithm for the context-free path query evaluation using the relational query semantics. According to Hellings, for a given graph $D = (V, E)$ and a grammar $G = (N, \Sigma, P)$ the context-free path query evaluation w.r.t. the relational query semantics reduces to a calculation of the context-free relations R_A. Thus, in this work, we focus on the calculation of conjunctive relations which are similar to the context-free relations.

Also, there is an algorithm [14] for path querying with linear conjunctive grammars and relational query semantics. This grammars have no more than one nonterminal in each conjunct of the rule. The possibility of creating an algorithm for path query evaluation w.r.t. conjunctive grammars of an arbitrary form is an open problem since the linear conjunctive grammars are known to be strictly less powerful than the arbitrary conjunctive grammars [8].

4 A path querying algorithm using conjunctive grammars

In this section, we show how the path querying using conjunctive grammars and relational query semantics can be reduced to the calculation of the matrix transitive closure. We propose an algorithm that calculates the over-approximation of all conjunctive relations R_A, since the query evaluation using the relational query semantics and conjunctive grammars is undecidable problem [5].

4.1 Reducing conjunctive path querying to transitive closure

In this section, we show how the over-approximation of all conjunctive relations R_A can be calculated by computing the transitive closure a^{conj}.

Let $G = (N, \Sigma, P)$ be a conjunctive grammar and $D = (V, E)$ be a graph. We number the nodes of the graph D from 0 to $(|V| - 1)$ and we associate the nodes with their numbers. We initialize $|V| \times |V|$ matrix b with $\emptyset$. Further, for every i and j we set $b_{i,j} = \{A_k \mid ((i,x,j) \in E) \wedge ((A_k \to x) \in P)\}$. Finally, we compute the conjunctive transitive closure $b^{conj} = b^{(1)} \cup b^{(2)} \cup \cdots$ where $b^{(i)} = b^{(i-1)} \cup (b^{(i-1)} \circ b^{(i-1)})$, $i \geq 2$ and $b^{(1)} = b$. For the conjunctive transitive closure b^{conj}, the following statements holds.

Lemma 1. *Let $D = (V, E)$ be a graph, let $G = (N, \Sigma, P)$ be a conjunctive grammar. Then for any i, j and for any non-terminal $A \in N$, if $(i, j) \in R_A$ and $i\pi j$, such that there is a derivation tree according to the string $l(\pi)$ and a conjunctive grammar $G_A = (N, \Sigma, P, A)$ of the height $h \leq k$ then $A \in b^{(k)}_{i,j}$.*

Proof. (Proof by Induction)

Basis: Show that the statement of the lemma holds for $k = 1$. For any i, j and for any non-terminal $A \in N$, if $(i, j) \in R_A$ and $i\pi j$, such that there is a derivation tree according to the string $l(\pi)$ and a conjunctive grammar $G_A = (N, \Sigma, P, A)$ of the height $h \leq 1$ then there is edge e from node i to node j and $(A \to x) \in P$ where $x = l(\pi)$. Therefore $A \in b^{(1)}_{i,j}$ and it has been shown that the statement of the lemma holds for $k = 1$.

Inductive step: Assume that the statement of the lemma holds for any $k \leq (p - 1)$ and show that it also holds for $k = p$ where $p \geq 2$. Let $(i, j) \in R_A$ and $i\pi j$, such that there is a derivation tree according to the string $l(\pi)$ and a conjunctive grammar $G_A = (N, \Sigma, P, A)$ of the height $h \leq p$.

Let $h < p$. Then by the inductive hypothesis $A \in b^{(p-1)}_{i,j}$. Since $b^{(p)} = b^{(p-1)} \cup (b^{(p-1)} \circ b^{(p-1)})$ then $A \in b^{(p)}_{i,j}$ and the statement of the lemma holds for $k = p$.

Let $h = p$. Let $A \to B_1C_1 \,\&\ldots\&\, B_mC_m$ be the rule corresponding to the root of the derivation tree from the assumption of the lemma. Therefore the heights of all subtrees corresponding to non-terminals $B_1, C_1, \ldots B_m, C_m$ are less than p. Then by the inductive hypothesis $B_x \in b^{(p-1)}_{i,t_x}$ and $C_x \in b^{(p-1)}_{t_x,j}$, for $x = 1 \ldots m$ and $t_x \in V$. Let d be a matrix that have subsets of $N \times N$ as elements, where $d_{i,j} = \bigcup_{t=1}^{n} b^{(p-1)}_{i,t} \times b^{(p-1)}_{t,j}$. Therefore $(B_x, C_x) \in d_{i,j}$, for $x = 1 \ldots m$. Since $b^{(p)} = b^{(p-1)} \cup (b^{(p-1)} \circ b^{(p-1)})$ and $(b^{(p-1)} \circ b^{(p-1)})_{i,j} = \{A \mid \exists (A \to B_1C_1 \,\&\ldots\&\, B_mC_m) \in P \text{ such that } (B_k, C_k) \in d_{i,j}\}$ then $A \in b^{(p)}_{i,j}$ and the statement of the lemma holds for $k = p$. This completes the proof of the lemma.

Theorem 1 *Let $D = (V, E)$ be a graph and let $G = (N, \Sigma, P)$ be a conjunctive grammar. Then for any i, j and for any non-terminal $A \in N$, if $(i, j) \in R_A$ then $A \in b^{conj}_{i,j}$.*

Proof. By the lemma 1, if $(i, j) \in R_A$ then $A \in b^{(k)}_{i,j}$ for some k, such that $i\pi j$ with a derivation tree according to the string $l(\pi)$ and a conjunctive grammar $G_A = (N, \Sigma, P, A)$ of the height $h \leq k$. Since the matrix $b^{conj} = b^{(1)} \cup b^{(2)} \cup \cdots$, then for any i, j and for any non-terminal $A \in N$, if $A \in b^{(k)}_{i,j}$ for some $k \geq 1$ then

$A \in b^{conj}_{i,j}$. Therefore, if $(i,j) \in R_A$ then $A \in b^{conj}_{i,j}$. This completes the proof of the theorem.

Thus, we show how the over-approximation of all conjunctive relations R_A can be calculated by computing the conjunctive transitive closure b^{conj} of the matrix b.

4.2 The algorithm

In this section we introduce an algorithm for calculating the conjunctive transitive closure b^{conj} which was discussed in Section 4.1.

The following algorithm takes on input a graph $D = (V, E)$ and a conjunctive grammar $G = (N, \Sigma, P)$.

Algorithm 1 Conjunctive recognizer for graphs

1: **function** conjunctiveGraphParsing(D, G)
2: $n \leftarrow$ a number of nodes in D
3: $E \leftarrow$ the directed edge-relation from D
4: $P \leftarrow$ a set of production rules in G
5: $T \leftarrow$ a matrix $n \times n$ in which each element is $\emptyset$
6: **for all** $(i, x, j) \in E$ **do** ▷ Matrix initialization
7: $T_{i,j} \leftarrow T_{i,j} \cup \{A \mid (A \to x) \in P\}$
8: **while** matrix T is changing **do**
9: $T \leftarrow T \cup (T \circ T)$ ▷ Transitive closure calculation
10: **return** T

Similar to the case of the context-free grammars we can show that the Algorithm 1 terminates in a finite number of steps. Since each element of the matrix T contains no more than $|N|$ non-terminals, then total number of non-terminals in the matrix T does not exceed $|V|^2|N|$. Therefore, the following theorem holds.

Theorem 2 *Let $D = (V, E)$ be a graph and let $G = (N, \Sigma, P)$ be a conjunctive grammar. Algorithm 1 terminates in a finite number of steps.*

Proof. It is sufficient to show, that the operation in line **9** of the Algorithm 1 changes the matrix T only finite number of times. Since this operation can only add non-terminals to some elements of the matrix T, but not remove them, it can change the matrix T no more than $|V|^2|N|$ times.

4.3 An example

In this section, we provide a step-by-step demonstration of the proposed algorithm for path querying using conjunctive grammars. The **example query** is based on the conjunctive grammar $G = (N, \Sigma, P)$ in binary normal form where:

- The set of non-terminals $N = \{S, A, B, C, D\}$.
- The set of terminals $\Sigma = \{a, b, c\}$.
- The set of production rules P is presented in Figure 1.

$$\begin{aligned}
&0 : S \to AB \;\&\; DC\\
&1 : A \to a\\
&2 : B \to BC\\
&3 : B \to b\\
&4 : C \to c\\
&5 : D \to AD\\
&6 : D \to b
\end{aligned}$$

Fig. 1: Production rules for the conjunctive example query grammar.

The conjunct AB generates the language $L_{AB} = \{abc^*\}$ and the conjunct DC generates the language $L_{DC} = \{a^*bc\}$. Thus, the language generated by the conjunctive grammar $G_S = (N, \Sigma, P, S)$ is $L(G_S) = L_{AB} \cap L_{DC} = \{abc\}$. We tun the query on a graph presented in Figure 2.

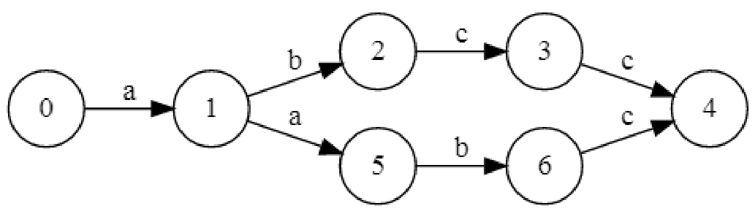

Fig. 2: An input graph for the conjunctive example query.

We provide a step-by-step demonstration of the work with the given graph D and grammar G of the Algorithm 1. After the matrix initialization in lines **6-7** of the Algorithm 1, we have a matrix T_0 presented in Figure 3.

Let T_i be the matrix T obtained after executing the loop in lines **8-9** of the Algorithm 1 i times. To compute the matrix T_1 we need to compute the matrix d where $d_{i,j} = \bigcup_{k=1}^{n} T_{0_{i,k}} \times T_{0_{i,k}}$. The matrix d for the first loop iteration is presented in Figure 4. The matrix $T_1 = T_1 = T_0 \cup (T_0 \circ T_0)$ is shown in Figure 5.

When the algorithm at some iteration finds new paths from the node i to the node j for all conjuncts of some production rule, then it adds nonterminal from the left side of this rule to the set $T_{i,j}$.

The calculation of the transitive closure is completed after k iterations when a fixpoint is reached: $T_{k-1} = T_k$. For this example, $k = 4$ since $T_4 = T_3$. The remaining iterations of computing the transitive closure are presented in Figure 6.

$$T_0 = \begin{pmatrix} \varnothing & \{A\} & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \{B,D\} & \varnothing & \varnothing & \{A\} & \varnothing \\ \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \{B,D\} \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \end{pmatrix}$$

Fig. 3: The initial matrix for the example query.

$$\begin{pmatrix} \varnothing & \varnothing & \{(A,B),(A,D)\} & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \{(B,C),(D,C)\} & \varnothing & \varnothing & \{(A,B),(A,D)\} \\ \varnothing & \varnothing & \varnothing & \varnothing & \{(C,C)\} & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{(B,C),(D,C)\} & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \end{pmatrix}$$

Fig. 4: The matrix d for the first loop iteration.

Thus, the result of the Algorithm 1 for the example query is the matrix $T_4 = T_3$. Now, after constructing the transitive closure, we can construct the over-approximations R'_A of the conjunctive relations R_A. These approximations for each non-terminal of the grammar G are presented in Figure 7.

This example demonstrates that it is not always possible to obtain an exact solution. For example, a pair of nodes $(0, 4)$ belongs to R'_S, although there is no path from the node 0 to the node 4, which forms a string derived from the nonterminal S (only the string abc can be derived from the nonterminal S). Extra pairs of nodes are added if there are different paths from the node i to the node j, which in summary correspond to all conjuncts of one production rule, but there is no path from the node i to the node j, which at the same time would correspond to all conjuncts of this rule. For example, for the conjuncts of the rule $S \to AB \ \& \ DC$, there is a path from the node 0 to the node 4 forming the string $abcc$, and there is also a path from the node 0 to the node 4 forming the string $aabc$. The first path corresponds to the conjunct AB, since the string $abcc$ belongs to the language $L_{AB} = \{abc^*\}$, and the second path corresponds to the conjunct DC, since the string $aabc$ belongs to the language $L_{DC} = \{a^*bc\}$. However, it is obvious that there is no path from the node 0 to the node 4, which forms the string abc.

$$T_1 = \begin{pmatrix} \varnothing & \{A\} & \{D\} & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \{B,D\} & \{B\} & \varnothing & \{A\} & \{D\} \\ \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{B\} & \varnothing & \{B,D\} \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \end{pmatrix}$$

Fig. 5: The initial matrix for the example query.

5 Evaluation

6 Conclusion and future work

References

1. Barrett, C., Jacob, R., Marathe, M.: Formal-language-constrained path problems. SIAM Journal on Computing **30**(3), 809–837 (2000)
2. Bastani, O., Anand, S., Aiken, A.: Specification inference using context-free language reachability. In: ACM SIGPLAN Notices. vol. 50, pp. 553–566. ACM (2015)
3. Grigorev, S., Ragozina, A.: Context-free path querying with structural representation of result. arXiv preprint arXiv:1612.08872 (2016)
4. Grune, D., Jacobs, C.J.H.: Parsing Techniques (Monographs in Computer Science). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)
5. Hellings, J.: Conjunctive context-free path queries (2014)
6. Hellings, J.: Querying for paths in graphs using context-free path queries. arXiv preprint arXiv:1502.02242 (2015)
7. Kasami, T.: An efficient recognition and syntaxanalysis algorithm for context-free languages. Tech. rep., DTIC Document (1965)
8. Okhotin, A.: Conjunctive and boolean grammars: the true general case of the context-free grammars. Computer Science Review **9**, 27–59 (2013)
9. Reps, T.: Program analysis via graph reachability. Information and software technology **40**(11), 701–726 (1998)
10. Reutter, J.L., Romero, M., Vardi, M.Y.: Regular queries on graph databases. Theory of Computing Systems **61**(1), 31–83 (2017)
11. Scott, E., Johnstone, A.: Gll parsing. Electronic Notes in Theoretical Computer Science **253**(7), 177–189 (2010)
12. Valiant, L.G.: General context-free recognition in less than cubic time. Journal of computer and system sciences **10**(2), 308–315 (1975)
13. Xu, G., Rountev, A., Sridharan, M.: Scaling cfl-reachability-based points-to analysis using context-sensitive must-not-alias analysis. In: ECOOP. vol. 9, pp. 98–122. Springer (2009)
14. Zhang, Q., Su, Z.: Context-sensitive data-dependence analysis via linear conjunctive language reachability. In: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages. pp. 344–358. ACM (2017)

$$T_2 = \begin{pmatrix} \varnothing & \{A\} & \{D\} & \{S\} & \varnothing & \varnothing & \{D\} \\ \varnothing & \varnothing & \{B,D\} & \{B\} & \{S,B\} & \{A\} & \{D\} \\ \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{B\} & \varnothing & \{B,D\} \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \end{pmatrix}$$

$$T_3 = \begin{pmatrix} \varnothing & \{A\} & \{D\} & \{S\} & \{S\} & \varnothing & \{D\} \\ \varnothing & \varnothing & \{B,D\} & \{B\} & \{S,B\} & \{A\} & \{D\} \\ \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{B\} & \varnothing & \{B,D\} \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \end{pmatrix}$$

$$T_4 = \begin{pmatrix} \varnothing & \{A\} & \{D\} & \{S\} & \{S\} & \varnothing & \{D\} \\ \varnothing & \varnothing & \{B,D\} & \{B\} & \{S,B\} & \{A\} & \{D\} \\ \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \varnothing & \varnothing & \varnothing & \varnothing & \{B\} & \varnothing & \{B,D\} \\ \varnothing & \varnothing & \varnothing & \varnothing & \{C\} & \varnothing & \varnothing \end{pmatrix}$$

Fig. 6: Remaining states of the matrix T.

$$R'_S = \{(0,3),(0,4),(1,4)\}, \tag{1}$$
$$R'_A = \{(0,1),(1,5)\}, \tag{2}$$
$$R'_B = \{(1,2),(1,3),(1,4),(5,4),(5,6)\}, \tag{3}$$
$$R'_C = \{(2,3),(3,4),(6,4)\}, \tag{4}$$
$$R'_D = \{(0,2),(0,6),(1,2),(1,6),(5,6)\}. \tag{5}$$

Fig. 7: The over-approximations of the conjunctive relations for the example query.