Efficiently Enumerating Answers to Ontology-Mediated Queries

Carsten Lutz
Institute of Computer Science
University of Leipzig
Leipzig, Germany
clu@informatik.uni-leipzig.de

Marcin Przybyłko
Institute of Computer Science
University of Leipzig
Leipzig, Germany
przybyl@informatik.uni-leipzig.de

ABSTRACT

We study the enumeration of answers to ontology-mediated queries (OMQs) where the ontology is a set of guarded TGDs or formulated in the description logic $\mathcal{ELI}$ and the query is a conjunctive query (CQ). In addition to the traditional notion of an answer, we propose and study two novel notions of partial answers that can take into account nulls generated by existential quantifiers in the ontology. Our main result is that enumeration of the traditional complete answers and of both kinds of partial answers is possible with linear-time preprocessing and constant delay for OMQs that are both acyclic and free-connex acyclic. We also provide partially matching lower bounds. Similar results are obtained for the related problems of testing a single answer in linear time and of testing multiple answers in constant time after linear time preprocessing. In both cases, the border between tractability and intractability is characterized by similar, but slightly different acyclicity properties.

CCS CONCEPTS

• **Theory of computation → Database query processing and optimization (theory)**; **Incomplete, inconsistent, and uncertain databases**.

KEYWORDS

ontology-mediated queries; tuple generating dependencies; description logic; enumeration; constant delay; partial answers

ACM Reference Format:
Carsten Lutz and Marcin Przybyłko. 2022. Efficiently Enumerating Answers to Ontology-Mediated Queries. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3517804.3524166

1 INTRODUCTION

In knowledge representation, ontologies are an important means for injecting domain knowledge into an application. In the context of databases, they give rise to ontology-mediated queries (OMQs) which enrich a traditional database query such as a conjunctive query (CQ) with an ontology. OMQs aim at querying incomplete data, using the domain knowledge provided by the ontology to derive additional answers. In addition, they may enrich the vocabulary available for query formulation with relation symbols that are not used explicitly in the data. Popular choices for the ontology language include (restricted forms of) tuple-generating dependencies (TDGs), also dubbed existential rules [5] and Datalog$^{\pm}$ [18], as well as various description logics [3].

The complexity of evaluating OMQs has been the subject of intense study, with a focus on *single-testing* as the mode of query evaluation: given an ontology-mediated query (OMQ) Q, a database D, and a candidate answer $\bar{a}$, decide whether $\bar{a} \in Q(D)$ [2, 6, 12, 14]. In many applications, however, it is not realistic to assume that a candidate answer is available. This has led database theoreticians and practitioners to investigate more relevant modes of query evaluation such as *enumeration*: given Q and D, generate all answers in $Q(D)$, one after the other and without repetition.

The first main aim of this paper is to initiate a study of efficiently enumerating answers to OMQs. We consider enumeration algorithms that have a preprocessing phase in which data structures are built that are used in the subsequent enumeration phase to produce the actual output. With 'efficient enumeration', we mean that preprocessing may only take time linear in $O(||D||)$ while the delay between two answers must be constant, that is, independent of D. One may impose the additional requirement that, in the enumeration phase, the algorithm may consume only a constant amount of memory on top of the data structures computed in the preprocessing phase. We follow [21, 35] and refer to the resulting enumeration complexity classes as $\mathsf{DelayC_{lin}}$ and CD∘Lin, the former admitting unrestricted (polynomial) memory consumption. Without ontologies, answer enumeration in CD∘Lin and in $\mathsf{DelayC_{lin}}$ has received significant attention [4, 11, 15, 20–22, 25, 26, 35], see also the survey [10]. A landmark result is that a CQ $q(\bar{x})$ admits enumeration in CD∘Lin if it is *acyclic* and *free-connex acyclic* where the former means that q has a join tree and the latter that the extension of q with an atom $R(\bar{x})$ that 'guards' the answer variables is acyclic [4]. Partially matching lower bounds pertain to self-join free CQs [4, 16].

The second aim of this paper is to introduce a novel notion of partial answers to OMQs. In the traditional *certain answers*, $\bar{a} \in Q(D)$ if and only if $\bar{a}$ is a tuple of constants from D such that $\bar{a} \in Q(I)$ for every model I of D and the ontology $\mathcal{O}$ used in Q. In contrast, a *partial answer* may contain, apart from constants from D, also the wildcard symbol '$*$' to indicate a constant that we know must exists, but whose identity is unknown. Such *labeled nulls* may be introduced by existential quantifiers in the ontology $\mathcal{O}$. To avoid redundancy as in the partial answers $(a, *)$ and (a, b), we are interested in *minimal* partial answers that cannot be 'improved' by replacing a wildcard with a constant from D while still remaining a partial answer. The following example illustrates that minimal

PODS '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9260-0/22/06…$15.00
https://doi.org/10.1145/3517804.3524166

partial answers provide useful information that is not conveyed by the traditional answers, from now called *complete answers*.

Example 1.1. Consider the ontology $\mathcal{O}$ that contains

$$\begin{aligned} \mathsf{Researcher}(x) &\rightarrow \exists y\, \mathsf{HasOffice}(x,y) \\ \mathsf{HasOffice}(x,y) &\rightarrow \mathsf{Office}(y) \\ \mathsf{Office}(x) &\rightarrow \exists y\, \mathsf{InBuilding}(x,y), \end{aligned}$$

and the CQ $q(x_1,x_2,x_3) = \mathsf{HasOffice}(x_1,x_2) \wedge \mathsf{InBuilding}(x_2,x_3)$ giving rise to the OMQ $Q(x_1,x_2,x_3)$. Take the following database D:

$$\mathsf{Researcher}(\mathsf{mary}) \quad \mathsf{Researcher}(\mathsf{john}) \quad \mathsf{Researcher}(\mathsf{mike})$$
$$\mathsf{HasOffice}(\mathsf{mary},\mathsf{room1}) \quad \mathsf{HasOffice}(\mathsf{john},\mathsf{room4})$$
$$\mathsf{InBuilding}(\mathsf{room1},\mathsf{main1})$$

The minimal partial answers to Q on D are

$$(\mathsf{mary},\mathsf{room1},\mathsf{main1}) \quad (\mathsf{john},\mathsf{room4},*) \quad (\mathsf{mike},*,*).$$

We also introduce and study *minimal partial answers with multiple wildcards* $*_1, *_2, \ldots$. Distinct occurences of the same wildcard in an answer indicate the same null, while different wildcards may or may not correspond to different nulls. Multiple wildcards may thus be viewed as adding equality on wildcards, but not inequality. We note that there are certain similarities between minimal partial answer to OMQs and answers to SPARQL queries with the 'optional' operator [9, 32], but also many dissimilarities.

The third aim of this paper is to study two problems for OMQs that are closely related to constant delay enumeration: single-testing in linear time (in data complexity) and *all-testing* in CD∘Lin or $\mathsf{DelayC_{lin}}$. Note that for Boolean queries, single-testing in linear time coincides with enumeration in CD∘Lin and in $\mathsf{DelayC_{lin}}$. An all-testing algorithm has a prepocessing phase followed by a testing phase where it repeatedly receives candidate answers $\bar{a}$ and returns 'yes' or 'no' depending on whether $\bar{a} \in Q(D)$ [10]. All-testing in $\mathsf{DelayC_{lin}}$ grants preprocessing time $O(||D||)$ while the time spent per test must be independent of D, and all-testing in CD∘Lin is defined accordingly.

An ontology-mediated query takes the form $Q(\bar{x}) = (\mathcal{O}, \mathbf{S}, q)$ where $\mathcal{O}$ is an ontology, $\mathbf{S}$ a schema for the databases on which Q is evaluated, and $q(\bar{x})$ a conjunctive query. In this paper, we consider ontologies that are sets of guarded tuple-generating dependencies (TGDs) or formulated in the description logic $\mathcal{ELI}$. We remind the reader that a TGD takes the form $\forall\bar{x}\forall\bar{y}\,(\phi(\bar{x},\bar{y}) \rightarrow \exists\bar{z}\,\psi(\bar{x},\bar{z}))$ where ϕ and ψ are CQs, and that it is *guarded* if ϕ has an atom that mentions all variables from $\bar{x}$ and $\bar{y}$. Up to normalization, an $\mathcal{ELI}$-ontology may be viewed as a finite set of guarded TGDs of a restricted form, using in particular only unary and binary relation symbols. Both guarded TGDs and $\mathcal{ELI}$ are natural and popular choices for the ontology language [3, 17, 19]. We use $(\mathbb{G}, \mathbb{CQ})$ to denote the language of all OMQs that use a set of guarded TGDs as the ontology and a CQ as the actual query, and likewise for $(\mathbb{ELI}, \mathbb{CQ})$ and $\mathcal{ELI}$-ontologies.

We next summarize our results. In Section 3, we start with showing that in $(\mathbb{G}, \mathbb{CQ})$, single-testing complete answers is in linear time for OMQs that are weakly acyclic. A CQ is *weakly acyclic* if it is acyclic after replacing the answer variables with constants and an OMQ is weakly acyclic if the CQ in it is; in what follows, we lift other properties of CQs to OMQs in the same way without further notice. Our proof relies on the construction of a 'query-directed' fragment of the chase and a reduction to the generation of minimal models of propositional Horn formulas. We also give a lower bound for OMQs from $(\mathbb{ELI}, \mathbb{CQ})$ that are self-join free: every such OMQ that admits single-testing in linear time is weakly acyclic unless the triangle conjecture from fine-grained complexity theory fails. This generalizes a result for the case of CQs without ontologies [16]. We observe that it is not easily possible to replace $\mathbb{ELI}$ by $\mathbb{G}$ in our lower bound as this would allow us to remove also 'self-join free' while it is open whether this is possible even in the case without ontologies. We also show that single-testing minimal partial answers with a single wildcard is in linear time for OMQs from $(\mathbb{G}, \mathbb{CQ})$ that are acyclic and that the same is true for multiple wildcards and acyclic OMQs from $(\mathbb{ELI}, \mathbb{CQ})$. We also observe that these (stronger) requirements cannot easily be relaxed.

In Section 4, we turn to enumeration and all-testing of complete answers. We first show that in $(\mathbb{G}, \mathbb{CQ})$, enumerating complete answers is in CD∘Lin for OMQs that are acyclic and free-connex acyclic while all-testing complete answers is in CD∘Lin for OMQs that are free-connex acyclic (but not necessarily acyclic). The proof again uses the careful chase construction and a reduction to the case without ontologies. The lower bound for single testing conditional on the triangle conjecture can be adapted to enumeration, with 'not weakly acyclic' replaced by 'not acyclic'. For enumeration, it thus remains to consider OMQs that are acyclic, but not free-connex acyclic. We show that for every self-join free OMQ from $(\mathbb{ELI}, \mathbb{CQ})$ that is acyclic, connected, and admits enumeration in CD∘Lin, the query is free-connex acyclic, unless sparse Boolean matrix multiplication (BMM) is possible in time linear in the size of the input plus the size of the ouput; this would imply a considerable advance in algorithm theory and currently seems to be out of reach. We also show that it is not possible to drop the requirement that the query is connected, which is not present in the corresponding lower bound for the case without ontologies [4, 10]. We prove a similar lower bound for all-testing complete answers, subject to a condition regarding non-sparse BMM. All mentioned lower bounds also apply to both kinds of partial answers.

In Section 5, we then prove that enumerating minimal partial answers with a single wildcard is in $\mathsf{DelayC_{lin}}$ for OMQs from $(\mathbb{G}, \mathbb{CQ})$ that are acyclic and free-connex acyclic. This is one of the main results of this paper, based on a non-trivial enumeration algorithm. Here, we only highlight two of its features. First, the algorithm precomputes certain data structures that describe 'excursions' that a homomorphism from q into the chase of D with $\mathcal{O}$ may make into the parts of the chase that has been generated by the existential quantifiers in the ontology. And second, it involves subtle sorting and pruning techniques to ensure that only *minimal* partial answers are output. We also observe that all-testing minimal partial answers is less well-behaved than enumeration as there is an OMQ $Q \in (\mathbb{ELI}, \mathbb{CQ})$ that is acyclic and free-connex acyclic, but for which all-testing is not in CD∘Lin unless the triangle conjecture fails.

Finally, Section 6 extends the upper bound from Section 5 to minimal partial answers with multiple wildcards. We first show that all-testing (not necessarily minimal!) partial answers with multiple wildcards is in $\mathsf{DelayC_{lin}}$ for OMQs that are acyclic and free-connex acyclic and then reduce enumeration of minimal partial answers

with multiple wildcards to this, combined with the enumeration algorithm of minimal partial answers with a single wildcard obtained in the previous section.

Most proof details are deferred to the long version [33].

2 PRELIMINARIES

Relational Databases. Fix countably infinite and disjoint sets of constants $\mathbf{C}$ and $\mathbf{N}$. We refer to the constants in $\mathbf{N}$ as *nulls*. A *schema* $\mathbf{S}$ is a set of relation symbols R with associated arity $\mathrm{ar}(R) \geq 0$. An $\mathbf{S}$-*fact* is an expression of the form $R(\bar{c})$, where $R \in \mathbf{S}$ and $\bar{c}$ is an $\mathrm{ar}(R)$-tuple of constants from $\mathbf{C} \cup \mathbf{N}$. An $\mathbf{S}$-*instance* is a set of $\mathbf{S}$-facts and an $\mathbf{S}$-*database* is a finite $\mathbf{S}$-instance that uses only constants from $\mathbf{C}$. We write $\mathrm{adom}(I)$ for the set of constants used in instance I. For a set $S \subseteq \mathbf{C} \cup \mathbf{N}$, $I_{|S}$ denotes the restriction of I to facts that mention only constants from S. A *homomorphism* from I to an instance J is a function $h : \mathrm{adom}(I) \to \mathrm{adom}(J)$ such that $R(h(\bar{c})) \in J$ for every $R(\bar{c}) \in I$. A set $S \subseteq \mathrm{adom}(I)$ is a *guarded set in* I if there is a fact $R(\bar{c}) \in I$ such that all constants from S are in $\bar{c}$. The *Gaifman graph* of a database D is the undirected graph with vertices $\mathrm{adom}(D)$ and an edge $\{c_1, c_2\}$ whenever c_1, c_2 co-occur in a fact in D.

Conjunctive Queries. A *term* is a variable or a constant from $\mathbf{C}$. A *conjunctive query* (CQ) $q(\bar{x})$ over a schema $\mathbf{S}$ takes the form $q(\bar{x}) \leftarrow \phi(\bar{x}, \bar{y})$ where $\bar{x}$ and $\bar{y}$ are tuples of variables, ϕ is a conjunction of *relational atoms* $R_i(\bar{t}_i)$ with $R_i \in \mathbf{S}$ and $\bar{t}_i$ a tuple of terms of length $\mathrm{ar}(R_i)$. The variables in $\bar{x}$ are the *answer variables* of q and the variables in $\bar{y}$ the *quantified variables*. With $\mathrm{var}(q)$, we denote the set of all variables in q and with $\mathrm{con}(q)$ the set of constants. Whenever convenient, we identify a conjunction of atoms with a set of atoms. The *arity* of q is defined as the number of its answer variables and q is Boolean if it is of arity 0. When we do not want to make $\phi(\bar{x}, \bar{y})$ explicit, we may denote $q(\bar{x}) \leftarrow \phi(\bar{x}, \bar{y})$ simply with $q(\bar{x})$. We say that $q(\bar{x})$ is *self-join free* if no relation symbol occurs in more than one atom in it. We write $\mathbb{CQ}$ for the class of CQs.

Every CQ $q(\bar{x})$ can be naturally seen as a database D_q, known as the *canonical database* of q, obtained by viewing variables as constants from $\mathbf{C}$. The *Gaifman graph* of q is that of D_q. A *homomorphism* h from q to an instance I is a homomorphism from D_q to I that is the identity on all constants that appear in q. A tuple $\bar{c} \in \mathrm{adom}(I)^{|\bar{x}|}$ is an *answer* to q on I if there is a homomorphism h from q to I with $h(\bar{x}) = \bar{c}$. The *evaluation of* $q(\bar{x})$ *on* I, denoted $q(I)$, is the set of all answers to q on I.

For a CQ q, but also for any other syntactic object q, we use $||q||$ to denote the number of symbols needed to write q as a word over a suitable alphabet.

Acyclic CQs. Let $q(\bar{x}) \leftarrow \phi(\bar{x}, \bar{y})$ be a CQ. A *join tree* for $q(\bar{x})$ is an undirected tree $T = (V, E)$ where V is the set of atoms in ϕ and for each variable $x \in \mathrm{var}(q)$, the set $\{\alpha \in V \mid x \text{ occurs in } \alpha\}$ is a connected subtree of T. Then $q(\bar{x})$ is *acyclic* if it has a join tree. Note that constants need not satisfy the connectedness condition imposed on variables. We say that $q(\bar{x})$ is *weakly acyclic* if q becomes acyclic after consistently replacing all answer variables with fresh constants. A CQ $q(\bar{x})$ is *free-connex acyclic* if adding an atom $R(\bar{x})$ that 'guards' the answer variables, where R is a relation symbol of arity $|\bar{x}|$, results in an acyclic CQ. Note that other authors have called a CQ q free-connex acyclic (or even just free-connex) if q is both acyclic and (in our sense) free-connex acyclic [10]. Acyclicity and free-connex acyclicity are independent properties, that is, neither of them implies the other. Each of them implies weak acyclicity while the converse is false. Figure 1 shows (the Gaifman graphs of) simple example CQs that illustrate the differences. Hollow nodes indicate quantified variables, ac stands for acyclic, fc for free-connex acyclic, and wac for weakly acyclic.

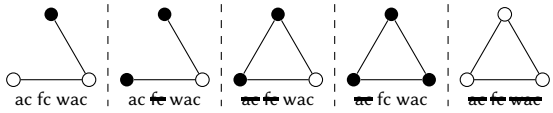

Figure 1: Different forms of acyclicity

TGDs, Guardedness, Chase. A *tuple-generating dependency* (TGD) T over $\mathbf{S}$ is a first-order sentence $\forall\bar{x}\forall\bar{y}\,(\phi(\bar{x}, \bar{y}) \to \exists\bar{z}\,\psi(\bar{x}, \bar{z}))$ such that $q_\phi(\bar{x}) \leftarrow \phi(\bar{x}, \bar{y})$ and $q_\psi(\bar{x}) \leftarrow \psi(\bar{x}, \bar{z})$ are CQs that do not contain constants. We call ϕ and ψ the *body* and *head* of T. The body may be the empty conjunction, i.e. logical truth, denoted by true. The variables in $\bar{x}$ are the *frontier variables*. For simplicity, we write T as $\phi(\bar{x}, \bar{y}) \to \exists\bar{z}\,\psi(\bar{x}, \bar{z})$. An instance I over $\mathbf{S}$ *satisfies* T, denoted $I \models T$, if $q_\phi(I) \subseteq q_\psi(I)$. It *satisfies* a set of TGDs $\mathcal{O}$, denoted $I \models \mathcal{O}$, if $I \models T$ for each $T \in \mathcal{O}$. We then also say that I is a *model* of $\mathcal{O}$. A TGD T is *guarded* if its body is true or contains a *guard atom* α that contains all variables in the body [17]. We write $\mathbb{TGD}$ to denote the class of all TGDs and $\mathbb{G}$ for the class of guarded TGDs.

The well-known chase procedure makes explicit in an instance the consequences of a set of TGDs [17, 28, 30, 34]. Let I be an instance and $\mathcal{O}$ be a set of TGDs. A TGD $T = \phi(\bar{x}, \bar{y}) \to \exists\bar{z}\,\psi(\bar{x}, \bar{z}) \in \mathcal{O}$ is *applicable* to a tuple $(\bar{c}, \bar{c}')$ of constants in I if $\phi(\bar{c}, \bar{c}') \subseteq I$. In this case, the *result of applying* T *in* I *at* $(\bar{c}, \bar{c}')$ is the instance $I \cup \{\psi(\bar{c}, \bar{c}'')\}$ where $\bar{c}''$ is the tuple obtained from $\bar{z}$ by simultaneously replacing each variable z with a fresh distinct null that does not occur in I. We refer to such an application as a *chase step*. A *chase sequence for* I *with* $\mathcal{O}$ is a sequence of instances $I_0, I_1, \ldots$ such that $I_0 = I$ and each I_{i+1} is the result of applying some TGD from $\mathcal{O}$ at some tuple $(\bar{c}, \bar{c}')$ of constants in I_i. The *result* of this chase sequence is the instance $J = \bigcup_{i \geq 0} I_i$. The chase sequence is *fair* if whenever a TGD $T \in \mathcal{O}$ is applicable to a tuple $(\bar{c}, \bar{c}')$ in some I_i, then this application is a chase step in the sequence. Fair chase sequences are oblivious in that a TGD is eventually applied whenever its body is satisfied, even if also its head is already satisfied. As a consequence, every fair chase sequence for I with $\mathcal{O}$ leads to the same result, up to isomorphism. We denote this result with $\mathsf{ch}_\mathcal{O}(I)$.

Ontology-Mediated Query, Description Logic. An *ontology* is a finite set of TGDs. An *ontology-mediated query (OMQ)* takes the form $Q = (\mathcal{O}, \mathbf{S}, q)$ where $\mathcal{O}$ is an ontology, $\mathbf{S}$ is a finite schema called the *data schema*, and q is a CQ. Both $\mathcal{O}$ and q can use symbols from $\mathbf{S}$, but also additional symbols, and in particular $\mathcal{O}$ can 'introduce' symbols to enrich the vocabulary available for querying. We assume w.l.o.g. that $\mathbf{S}$ contains only relation symbols that occur in $\mathcal{O}$ or q. The *arity* of Q is defined as the arity of q. We write $Q(\bar{x})$ to emphasize that the answer variables of q are $\bar{x}$ and say that Q is *acyclic* if q is and likewise for *weakly acyclic*, *free-connex acyclic*, *self-join free*, and so on.

A tuple $\bar{c} \in \mathrm{adom}(D)^{|\bar{x}|}$ is a *(certain) answer* to Q on D if $\bar{c} \in q(I)$ for every model I of $\mathcal{O}$ with $I \supseteq D$. The *evaluation of* $Q(\bar{x})$ over D, denoted $Q(D)$, is the set of all answers to Q over D. Importantly, $Q(D) = q(\mathrm{ch}_{\mathcal{O}}(D))$ for every OMQ $Q = (\mathcal{O}, \mathbf{S}, q)$ and $\mathbf{S}$-database D. When convenient, we may write $D \cup \mathcal{O} \models q(\bar{c})$ in place of $\bar{c} \in Q(D)$. We say that Q is *empty* if $Q(D) = \emptyset$ for all $\mathbf{S}$-databases D.

Let us remark that a CQ q can be semantically acyclic in the sense that it is equivalent to an acyclic CQ, but not acyclic itself [8, 24]. It is known that this is the case if and only if the homomorphism core of q is acyclic. An OMQ can be semantically acyclic (in the same sense) even if the homomorphism core of the CQ in it is not acyclic, that is, the ontology has an impact on semantic acyclicity; see [6, 7] for very similar effects that pertain to bounded treewidth. Since we are concerned with data complexity in this article, we can simply replace an OMQ with any equivalent one and thus w.l.o.g. refrain from considering semantic acyclicity.

We next introduce the widely known description logic $\mathcal{ELI}$ [3]. Traditionally, description logics come with their own variable-free syntax. Here, we introduce $\mathcal{ELI}$ using TGD syntax. A guarded TGD $\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z})$ is an $\mathcal{ELI}$ *TGD* if it uses only unary and binary relation symbols, has only a single frontier variable, contains no reflexive loops and multi-edges in body or head, and has a head that is acyclic and connected. Note that the original definition of $\mathcal{ELI}$ is more liberal in that it restricts the body in the same way as the head in our definition, thus encompassing also unguarded TGDs. However, the restricted form used here can be attained by syntactic normalization [3]. Since the normalization of an ontology inside an OMQ does not affect query answers, all results in this paper apply also to the more liberal definition of $\mathcal{ELI}$. We use $\mathbb{ELI}$ to denote the set of all $\mathcal{ELI}$ TGDs.

An *OMQ language* is a class of OMQs. For a class of TGDs $\mathbb{C}$ and a class of CQs $\mathbb{Q}$, we write $(\mathbb{C}, \mathbb{Q})$ to denote the OMQ language that consists of all OMQs $(\mathcal{O}, \mathbf{S}, q)$ where $\mathcal{O}$ is a set of TGDs from $\mathbb{C}$ and $q \in \mathbb{Q}$. For example, we may write $(\mathbb{G}, \mathbb{CQ})$ and $(\mathbb{ELI}, \mathbb{CQ})$.

Let $Q_i(\bar{x}) = (\mathcal{O}_i, \mathbf{S}, q_i)$ for $i \in \{1, 2\}$. Then OMQ Q_1 is *contained* in OMQ Q_2, written $Q_1 \subseteq Q_2$, if $Q_1(D) \subseteq Q_2(D)$ for every $\mathbf{S}$-database D. Moreover, Q_1 and Q_2 are *equivalent*, written $Q_1 \equiv Q_2$, if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$.

Machine Model. As our computational model, we use RAMs under the uniform cost model [23], see [29] for a formalization. Such a RAM has a one-way read-only input tape, a write-only output tape, and an unbounded number of registers that store non-negative integers of $O(\log n)$ bits, n the input size; this is called a DRAM in [29], used there to define the complexity class DLINEAR. Adding, subtracting, and comparing the values of two registers as well as bit shift takes time $O(1)$. On a DRAM, sorting is possible in linear time and we can use and access lookup tables indexed by constants from $\mathrm{adom}(D)$ or by tuples of constants of length $O(1)$ [29]. This model is standard in the context of constant delay enumeration, see [4, 10, 20, 35] and the long version for more details.

Modes of Query Evaluation. *Single-testing* means to decide, given an OMQ $Q(\bar{x}) = (\mathcal{O}, \mathbf{S}, q)$, an $\mathbf{S}$-database D, and an answer candidate $\bar{c} \in \mathrm{adom}(D)^{|\bar{x}|}$, whether $\bar{c} \in Q(D)$. We generally consider data complexity, where the OMQ Q is fixed and thus of constant size and the only remaining inputs are D and $\bar{c}$.

An *enumeration algorithm* for a class of OMQs $\mathbb{C}$ is given as inputs an OMQ $Q(\bar{x}) = (\mathcal{O}, \mathbf{S}, q) \in \mathbb{C}$ and an $\mathbf{S}$-database D. In the *preprocessing phase*, it may produce data structures, but no output. In the subsequent *enumeration phase*, it enumerates all tuples from $Q(D)$, without repetition, followed by an end of enumeration signal. An *all-testing algorithm* for $\mathbb{C}$ is defined similarly. It takes the same two inputs, and has the same preprocessing phase, followed by a *testing phase* where it repeatedly receives tuples $\bar{a} \in \mathrm{adom}(D)^{|\bar{x}|}$ and returns 'yes' or 'no' depending on whether $\bar{a} \in Q(D)$.

Let $(\mathbb{L}, \mathbb{Q})$ be an OMQ language. We say that *answer enumeration for* $(\mathbb{L}, \mathbb{Q})$ *is possible with linear preprocessing and constant delay*, or *in* $\mathsf{DelayC_{lin}}$ for short, if there is an enumeration algorithm for $(\mathbb{L}, \mathbb{Q})$ in which preprocessing takes time $f(||Q||) \cdot O(||D||)$, f a computable function, while the delay between the output of two consecutive answers depends only on $||Q||$, but not on $||D||$. Enumeration in CD∘Lin is defined likewise, except that in the enumeration phase, the algorithm may consume only a constant amount of memory. Accessing the data structures computed in the preprocessing phase does not count as memory usage. It is not clear whether $\mathsf{DelayC_{lin}}$ and CD∘Lin coincide or not, see e.g. [31]. The definition of $\mathsf{DelayC_{lin}}$ and CD∘Lin for all-testing is analogous, except that the enumeration delay is replaced with the time needed for testing.

Partial Answers. We first introduce partial answers with a single wildcard symbol '$*$' (that is not in $\mathbf{C} \cup \mathbf{N}$). A *wildcard tuple* for an instance I takes the form $(c_1, \ldots, c_n) \in (\mathrm{adom}(I) \cup \{*\})^n$, $n \geq 0$. For wildcard tuples $\bar{c} = (c_1, \ldots, c_n)$ and $\bar{c}' = (c'_1, \ldots, c'_n)$, we write $\bar{c} \leq \bar{c}'$ if $c'_i \in \{c_i, *\}$ for $1 \leq i \leq n$. Moreover, $\bar{c} \prec \bar{c}'$ if $\bar{c} \leq \bar{c}'$ and $\bar{c} \neq \bar{c}'$. For example, $(a, b) \prec (a, *)$ and $(a, *) \prec (*, *)$. Informally, $\bar{c} \prec \bar{c}'$ expresses that tuple $\bar{c}$ is preferred over tuple $\bar{c}'$ as it carries more information. A *partial answer* to OMQ $Q(\bar{x}) = (\mathcal{O}, \mathbf{S}, q)$ on $\mathbf{S}$-database D is a wildcard tuple $\bar{c}$ for D of length $|\bar{x}|$ such that for each model I of $\mathcal{O}$ with $I \supseteq D$, there is a $\bar{c}' \in q(I)$ such that $\bar{c}' \leq \bar{c}$. Note that some positions in $\bar{c}'$ may contain constants from $\mathrm{adom}(I) \setminus \mathrm{adom}(D)$, and that the corresponding position in $\bar{c}$ must then have a wildcard. A partial answer $\bar{c}$ to Q on $\mathbf{S}$-database D is a *minimal partial answer* if there is no partial answer $\bar{c}'$ to Q on D with $\bar{c}' \prec \bar{c}$. The *partial evaluation of* $Q(\bar{x})$ on D, denoted $Q(D)^*$, is the set of all minimal partial answers to Q on D. Note that $Q(D) \subseteq Q(D)^*$. An illustrating example is provided in Section 1.

Minimal partial answers may provide valuable information not captured by complete answers. However, one might argue that complete answers are more important than minimal partial answers that contain a wildcard, and should thus be output first by an enumeration algorithm. We observe that this is always possible if we are interested in $\mathsf{DelayC_{lin}}$ (whereas it is not clear whether an analogous statement for CD∘Lin holds).

Proposition 2.1. *Let $Q \in (\mathbb{TGD}, \mathbb{CQ})$. If minimal partial answers to Q can be enumerated in $\mathsf{DelayC_{lin}}$ and the same is true for complete answers, then there is a $\mathsf{DelayC_{lin}}$ enumeration algorithm for minimal partial answers to Q that produces the complete answers first.*

We next introduce partial answers with multiple wildcards. Fix a countably infinite set of *wildcards* $\mathcal{W} = \{*_1, *_2, \ldots\}$ (that are not in $\mathbf{C} \cup \mathbf{N}$). A *multi-wildcard tuple* for an instance I is a tuple $(c_1, \ldots, c_n) \in (\mathrm{adom}(I) \cup \mathcal{W})^n$, $n \geq 0$, such that if $c_i = *_j$ with $j > 1$, then there is an $i' < i$ with $c_{i'} = *_{j-1}$. Examples for

multi-wildcard tuples are $(*_1, *_2)$ and $(a, *_1, b, a, *_2, *_1, *_2)$ and a non-example is $(*_2, *_1)$. Occurrences of the same wildcard represent occurrences of the same null while different wildcards represent nulls that may or may not be different. For multi-wildcard tuples $\bar{c} = (c_1, \ldots, c_n)$ and $\bar{c}' = (c'_1, \ldots, c'_n)$, we write $\bar{c} \preceq \bar{c}'$ if

(1) $c_i = c'_i$ or $\mathcal{W} \not\ni c_i \neq c'_i \in \mathcal{W}$ for $1 \leq i \leq n$ and
(2) $c'_i = c'_j$ implies $c_i = c_j$ for $1 \leq i, j \leq n$.

Moreover, $\bar{c} \prec \bar{c}'$ if $\bar{c} \preceq \bar{c}'$ and $\bar{c} \neq \bar{c}'$. For example, $(*_1, a) \prec (*_1, *_2)$ and $(a, *_1, *_2, *_1) \prec (a, *_1, *_2, *_3)$. *Partial answers with multi-wildcards* and *minimal partial answers with multi-wildcards* are defined in exact analogy with (minimal) partial answers, but using multi-wildcard tuples in place of wildcard tuples. The *partial evaluation of $Q(\bar{x})$ with multi-wildcards* on D, denoted $Q(D)^{\mathcal{W}}$, is the set of all minimal partial answers with multi-wildcards to Q on D.

Example 2.2. Reconsider the ontology OMQ $Q = (\mathcal{O}, \mathbf{S}, q)$ and database D from Example 1.1. Then $Q(D)^{\mathcal{W}}$ contains the tuples

$$(\mathsf{mary}, \mathsf{room1}, \mathsf{main1}) \quad (\mathsf{john}, \mathsf{room4}, *_1) \quad (\mathsf{mike}, *_1, *_2).$$

Let the ontology $\mathcal{O}'$ be obtained from $\mathcal{O}$ by adding

$$\mathsf{Prof}(x) \wedge \mathsf{HasOffice}(x, y) \quad \rightarrow \quad \mathsf{LargeOffice}(y)$$

and $\mathbf{S}'$ from $\mathbf{S}$ by adding LargeOffice, consider the CQ

$$q'(x_1, x_2, x_3, x_4) \quad = \quad \mathsf{HasOffice}(x_1, x_2) \wedge \mathsf{LargeOffice}(x_2) \wedge \mathsf{HasOffice}(x_1, x_3) \wedge \mathsf{InBuilding}(x_3, x_4),$$

and let $Q' = (\mathcal{O}', \mathbf{S}', q')$. Moreover, let D' be D extended with fact

$$\mathsf{Prof}(\mathsf{mike}).$$

Then $Q'(D')^{\mathcal{W}}$ contains, e.g., the tuple $(\mathsf{mike}, *_1, *_1, *_2)$, but not the tuple $(\mathsf{mike}, *_1, *_2, *_3)$ which is a partial answer, but not minimal.

Finally, let the ontology $\mathcal{O}''$ be obtained from $\mathcal{O}$ by adding

$$\mathsf{OfficeMate}(x, y) \quad \rightarrow \quad \exists z\, \mathsf{HasOffice}(x, z) \wedge \mathsf{HasOffice}(y, z)$$

and $\mathbf{S}''$ from $\mathbf{S}$ by adding OfficeMate, consider the CQ

$$q''(x_1, x_2, x_3, x_4) \quad = \quad \exists y\, \mathsf{Hasoffice}(x_1, x_3) \wedge \mathsf{Hasoffice}(x_2, x_4) \wedge \mathsf{InBuilding}(x_3, y) \wedge \mathsf{InBuilding}(x_4, y),$$

and set $Q''(x_1, x_2, x_3, x_4) = (\mathcal{O}'', \mathbf{S}'', q'')$. Moreover, let D'' be D extended with fact

$$\mathsf{OfficeMate}(\mathsf{mary}, \mathsf{mike}).$$

$Q''(D'')^{\mathcal{W}}$ contains, e.g., the tuple $(\mathsf{mary}, \mathsf{mike}, *_1, *_1)$.

Minimal partial answers can equivalently be defined in terms of the chase. Let $q(\bar{x})$ be a CQ and I an instance, possibly containing nulls. For an answer $\bar{a} \in q(I)$, we use $\bar{a}^*_{\mathbf{N}}$ to denote the unique wildcard tuple for I obtained from $\bar{a}$ by replacing all nulls with '$*$'. We call such an $\bar{a}^*_{\mathbf{N}}$ a *partial answer* to q on I and say that it is a *minimal* partial answer if there is no $\bar{b} \in q(I)$ with $\bar{b}^*_{\mathbf{N}} \prec \bar{a}^*_{\mathbf{N}}$. We use $q(I)^*_{\mathbf{N}}$ to denote the set of minimal partial answers to q on I. Similarly, we use $\bar{a}^{\mathcal{W}}_{\mathbf{N}}$ to denote the unique multi-wildcard tuple for I obtained by consistently replacing all nulls with wildcards from $\mathcal{W} = \{*_1, *_2, \ldots\}$. We then define *minimal partial answer with multi-wildcards* to q on I, denoted $q(D)^{\mathcal{W}}_{\mathbf{N}}$, in the expected way.

Lemma 2.3. *Let $Q(\bar{x}) = (\mathcal{O}, \mathbf{S}, q) \in (\mathbb{TGD}, \mathbb{CQ})$ and D be an $\mathbf{S}$-database. Then $Q(D)^* = q(\mathsf{ch}_{\mathcal{O}}(D))^*_{\mathbf{N}}$ and $Q(D)^{\mathcal{W}} = q(\mathsf{ch}_{\mathcal{O}}(D))^{\mathcal{W}}_{\mathbf{N}}$.*

3 SINGLE-TESTING

We consider the limits of single-testing in linear time for the OMQ languages $(\mathbb{G}, \mathbb{CQ})$ and $(\mathbb{ELI}, \mathbb{CQ})$. For complete answers, we establish a close link to weak acyclicity while minimal partial answers with a single wildcard are linked (in a more loose way) to acyclicity. The latter is also achieved for minimal partial answers with multi-wildcards, but only when the ontology is from $\mathbb{ELI}$. To the best of our knowledge, these are the first results on linear time single-testing for ontology-mediated queries. Existing algorithms from the literature seem to require at least quadratic time (although authors typically do not analyse the degree of the polynomial explicitly).

Theorem 3.1. *Single-testing is in linear time for*

(1) *weakly acyclic OMQs from $(\mathbb{G}, \mathbb{CQ})$ in the case of complete answers;*
(2) *acyclic OMQs from $(\mathbb{G}, \mathbb{CQ})$ in the case of minimal partial answers with single wildcards;*
(3) *acyclic OMQs from $(\mathbb{ELI}, \mathbb{CQ})$ in the case of minimal partial answers with multi-wildcards.*

To prove Theorem 3.1, we first show that for every OMQ $Q(\bar{x}) = (\mathcal{O}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{CQ})$ and $\mathbf{S}$-database D, one can compute in time linear in $||D||$ a (finite!) database $\mathsf{ch}^q_{\mathcal{O}}(D)$ that enjoys all properties of the chase $\mathsf{ch}_{\mathcal{O}}(D)$ which are important for enumerating answers to Q, both complete and partial. Informally, $\mathsf{ch}^q_{\mathcal{O}}(D)$ contains only those parts of $\mathsf{ch}_{\mathcal{O}}(D)$ that are 'relevant to q'. We refer to $\mathsf{ch}^q_{\mathcal{O}}(D)$ as the *query-directed chase*, similar constructions have been used e.g. in [6, 13].

Let $\mathsf{cl}(Q)$ denote the set of CQs that are connected and use only relation symbols that occur in $\mathcal{O}$, no constants, and only variables from a fixed set V whose cardinality is the maximum of $|\mathsf{var}(q)|$ and the arities of relation symbols in $\mathcal{O}$. Note that the CQs in $\mathsf{cl}(Q)$ may have any arity, including zero, and that the number of CQs in $\mathsf{cl}(Q)$ is independent of D. The database $\mathsf{ch}^q_{\mathcal{O}}(D)$ is obtained from D by adding, for every CQ $p(\bar{y}) \in \mathsf{cl}(Q)$ and every $\bar{c} \in \mathsf{adom}(D)^{|\bar{y}|}$ such that $D \cup \mathcal{O} \models p(\bar{c})$ and the constants in $\bar{c}$ constitute a guarded set in D, a copy of D_p that uses the constants in $\bar{c}$ in place of the answer variables $\bar{y}$ of p and only fresh constants otherwise.

Lemma 3.2. *Let $Q(\bar{x}) = (\mathcal{O}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{CQ})$ and D be an $\mathbf{S}$-database. Then $Q(D) = q(\mathsf{ch}^q_{\mathcal{O}}(D)) \cap \mathsf{adom}(D)^{|\bar{x}|}$, $Q(D)^* = q(\mathsf{ch}^q_{\mathcal{O}}(D))^*_{\mathbf{N}}$, and $Q(D)^{\mathcal{W}} = q(\mathsf{ch}^q_{\mathcal{O}}(D))^{\mathcal{W}}_{\mathbf{N}}$.*

As announced, the query-directed chase can be computed in linear time. Q is not required to be acyclic for this to hold.

Proposition 3.3. *Let $Q(\bar{x}) = (\mathcal{O}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{CQ})$ and let D be an $\mathbf{S}$-database. Then $\mathsf{ch}^q_{\mathcal{O}}(D)$ can be computed in time linear in $||D||$, more precisely in time $2^{2^{O(||Q||^2)}} \cdot ||D||$.*

To prove Proposition 3.3, we derive from D and Q a satisfiable Horn formula θ, make use of the fact that a minimal model of θ can be computed in linear time [27], and then read off $\mathsf{ch}^q_{\mathcal{O}}(D)$ from the minimal model. We are not aware that such an approach has been used before.

For Point (1) of Theorem 3.1, we have to check whether $\bar{c} \in Q(D)$ which can now be done in linear time a straightforward way. First compute $\mathsf{ch}^q_{\mathcal{O}}(D)$. Then replace the answer variables in q by the

constants from $\bar{c}$, turning the weakly acyclic q into an acyclic CQ. Finally, use an existing procedure such as Yannakakis' algorithm to single-test the resulting CQ in linear time [36]. Points (2) and (3) of Theorem 3.1 are proved by a (Turing) reduction to the case of complete answers. Details are provided in the long version.

We next prove a lower bound that partially matches Theorem 3.1. As in the case without ontologies, we do not obtain a full dichotomy as the lower bound only applies to queries that are self-join free. In addition (and related to this), it only applies to OMQs where the ontology is formulated in the subclass $\mathbb{ELI}$ of $\mathbb{G}$. The lower bound is conditional on the triangle conjecture, which we formulate next. *Triangle detection* is the problem to decide, given an undirected graph $G = (V, E)$ as a list of edges, whether G contains a 3-clique. The triangle conjecture from fine-grained complexity theory [1] states that triangle detection cannot be solved in linear time.

Theorem 3.4. *Let $Q \in (\mathbb{ELI}, \mathbb{CQ})$ be non-empty and self-join free. If Q is not weakly acyclic, single-testing complete answers to Q is not in linear time unless the triangle conjecture fails. The same is true for minimal partial answers and minimal partial answers with multiple wildcards.*

The proof of Theorem 3.4 is an adaptation of the construction given in [10, 16] where no ontologies are considered. The challenge is to deal with the ontology and the fact that the ontology may contain relation symbols that are not admitted in the database. We address this by modifying the database construction from [10, 16] so that every constant c comes with fact $A(c)$ for every unary relation symbol $A \in \mathbf{S}$ and has an incoming and an outgoing R-edge for every binary relation symbol $R \in \mathbf{S}$. Informally, this ensures that everything that could possibly be implied by the ontology is indeed implied. Self-join freeness is important for this approach to work.

While it would be desirable to replace $\mathbb{ELI}$ with $\mathbb{G}$ in Theorem 3.4, this seems hard to achieve as it would also allow us to remove 'self-join free' from that theorem. Even in the case without ontologies, it is currently not known whether this is possible.

Example 3.5. Let $Q(\bar{x}) = (\mathcal{O}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{CQ})$ and let $Q' = (\mathcal{O}', \mathbf{S}, q')$ be the OMQ that can be obtained from Q as follows: consider every atom $R(\bar{z})$ in q, replace it with $R_{\bar{z}}(\bar{z})$ where $R_{\bar{z}}$ is a fresh relation symbol of the same arity as R, and add to $\mathcal{O}$ the TGDs

$$R(\bar{x}) \to R_{\bar{z}}(\bar{x}) \text{ and } R_{\bar{z}}(\bar{x}) \to R(\bar{x})$$

where $\bar{x}$ is a tuple of $\mathrm{ar}(R)$ distinct variables. Then $Q \equiv Q'$, and Q' is self-join free. Moreover, Q' is weakly acyclic if and only if Q is.

More examples regarding Theorem 3.4 are given in the long version. We close with noting that the prerequisites given in Theorem 3.1 for the case of minimal partial answers cannot easily be relaxed.

Theorem 3.6. *(1) There is a weakly acyclic OMQ $Q \in (\mathbb{ELI}, \mathbb{CQ})$ for which single-testing minimal partial answers is not in linear time unless the triangle conjecture fails and (2) an acyclic OMQ $Q \in (\mathbb{G}, \mathbb{CQ})$ for which single-testing minimal partial answers with multi-wildcards is not in linear time unless the triangle conjecture fails.*

4 ENUMERATION AND ALL-TESTING: COMPLETE ANSWERS

We consider the limits of enumeration and all-testing of complete answers with constant delay for the OMQ languages $(\mathbb{G}, \mathbb{CQ})$ and $(\mathbb{ELI}, \mathbb{CQ})$. While enumeration is linked to the combination of acyclicity and free-connex acyclicity, we link all-testing to free-connex acyclicity only. In the lower bounds, we also consider minimal partial answers and minimal partial answers with multiple wildcards. We start with the upper bounds.

Theorem 4.1. *In $(\mathbb{G}, \mathbb{CQ})$,*

(1) *enumerating complete answers is in* CD∘Lin *for OMQs that are acyclic and free-connex acyclic;*
(2) *all-testing complete answers is in* CD∘Lin *for OMQs that are free-connex acyclic.*

Recall that for a CQ q to be free-connex acyclic, we do *not* require q to be acyclic. Thus, the requirement for all-testing in Theorem 4.1 is significantly weaker than that for enumeration and embraces, for example, every OMQ in which the CQ is full, that is, has no quantified variables. The proof of Point (1) of Theorem 4.1 uses the query-directed chase also employed in Section 3 and a reduction to the CD∘Lin enumeration of answers to CQs (without ontologies) that are acyclic and free-connex acyclic [4]. Point (2) can be proved in the same way using the following observation which, to our knowledge, is novel.

Proposition 4.2. *For CQs (without ontologies) that are free-connex acyclic, all-testing is in* CD∘Lin.

To prove Proposition 4.2, we decompose the given CQ into CQs that are acyclic and free-connex acyclic, and then use CD∘Lin all-testing algorithms for those component CQs in parallel. In the long version, we give a matching lower bound for self-join free CQs.

We next give lower bounds that partially match Theorem 4.1, starting with the requirement in Point (1) of Theorem 4.1 that OMQs must be acyclic. The following is a consequence of Theorem 3.4.

Theorem 4.3. *Let $Q \in (\mathbb{ELI}, \mathbb{CQ})$ be non-empty, and self-join free. If Q is not acyclic, then enumerating complete answers to Q is not in* $\mathsf{DelayC_{lin}}$ *unless the triangle conjecture fails. The same is true for minimal partial answers and for minimal partial answers with multiple wildcards.*

In Theorem 4.3 and all other lower bounds stated in this section, $\mathbb{ELI}$ cannot easily be replaced by $\mathbb{G}$, see Example 3.5.

Staying with the requirements of Point (1) of Theorem 4.2, we next consider queries that are acyclic, but not free-connex acyclic. The lower bound that we establish is conditional on an assumption regarding the problem of Sparse Boolean matrix multiplication. A Boolean $n \times n$ matrix is a function $M : [n]^2 \to \{0, 1\}$ where $[n]$ denotes the set $\{1, \ldots, n\}$. The *product* of two Boolean $n \times n$ matrices M_1, M_2 is the Boolean $n \times n$ matrix $M_1 M_2 := \sum_{c=1}^{n} M_1(a, c) \cdot M_2(c, b)$ where sum and product are interpreted over the Boolean semiring. In (non-sparse) *Boolean matrix multiplication (BMM)*, one wants to compute $M_1 M_2$ given M_1 and M_2 as $n \times n$ arrays. In *sparse Boolean matrix multiplication (spBMM)*, input and output matrices M are represented as lists of pairs (a, b) with $M(a, b) = 1$. Our lower bound is conditional on the assumption that spBMM is not possible in

time $O(|M_1| + |M_2| + |M_1M_2|)$, that is, in time linear in the size of the input and the output (represented as lists). While it is not ruled out that such a running time can be achieved, this would require dramatic progress in algorithm theory. Informally, the conditioning on spBMM should be read as 'currently out of reach'.

Theorem 4.4. *Let $Q = (\mathcal{O}, \mathbf{S}, q) \in (\mathbb{ELI}, \mathbb{CQ})$ be acyclic, non-empty, self-join free, and connected. If Q is not free-connex acyclic, then enumerating complete answers to Q is not in* $\mathsf{DelayC_{lin}}$ *unless spBMM is possible in time $O(|M_1| + |M_2| + |M_1M_2|)$. The same is true for minimal partial answers and for minimal partial answers with multiple wildcards.*

There is a corresponding lower bound for CQs without ontologies, first proved conditional on the assumption that Boolean $n \times n$ matrices cannot be multiplied in time $O(n^2)$ [4] and then improved to the condition on spBMM used in Theorem 4.4 in [10]. To prove Theorem 4.4, we again have to deal with the fact that the ontology may contain relation symbols that are not admitted in the database. Here, this is done by first manipulating the input matrices M_1 and M_2 in a suitable way. Note that we require Q to be connected while this is not a precondition in the case without ontologies [10]. The following proposition shows that we cannot drop connectedness.

Proposition 4.5. *There is an OMQ $Q \in (\mathbb{ELI}, \mathbb{CQ})$ that is acyclic, non-empty, self-join free, but neither free-connex acyclic nor connected, such that complete answers to Q can be enumerated in* $\mathsf{DelayC_{lin}}$.

We next address the requirement in Point (2) of Theorem 4.2 that OMQs must be free-connex acyclic.

Theorem 4.6. *Let $Q \in (\mathbb{ELI}, \mathbb{CQ})$ be non-empty and self-join free. If Q is not free-connex acyclic, then all-testing complete answers for Q is not in linear time unless the triangle conjecture fails or Boolean $n \times n$ matrices can be multiplied in time $O(n^2)$. The same is true for minimal partial answers and minimal partial answers with multiple wildcards.*

Note that Theorem 4.6 refers to the non-sparse version of BMM and that spBMM in time $O(|M_1| + |M_2| + |M_1M_2|)$ implies BMM in time $O(n^2)$ while the converse is unknown.

5 ENUMERATION WITH SINGLE WILDCARD

The main aim of this section is to prove that it is possible to enumerate in $\mathsf{DelayC_{lin}}$ the minimal partial answers with a single wildcard to OMQs from $(\mathbb{G}, \mathbb{CQ})$ that are acyclic and free-connex acyclic. Thus, minimal partial answers are almost as well-behaved as complete answers, except that for the former it remains open whether enumeration is also possible in CD∘Lin. We start, however, with observing that all-testing of minimal partial answers is less well-behaved. The following should be contrasted with Point (2) of Theorem 4.1.

Theorem 5.1. *There is an OMQ $Q \in (\mathbb{ELI}, \mathbb{CQ})$ that is acyclic and free-connex acyclic such that all-testing minimal partial answers to Q is not in* $\mathsf{DelayC_{lin}}$ *unless the triangle conjecture fails. The same is true for minimal partial answers with multiple wildcards.*

Intuitively, all-testing of minimal partial answers is difficult because a single positive test for an answer that contains wildcards may imply a negative test for polynomially many complete answers. This is not a problem in enumeration where the 'problematic' wildcard answers will be output late and thus cannot be tested in linear time.

We now turn to the main result of this section.

Theorem 5.2. *Enumerating minimal partial answers is in* $\mathsf{DelayC_{lin}}$ *for OMQs from $(\mathbb{G}, \mathbb{CQ})$ that are acyclic and free-connex acyclic.*

In the rest of this section, we prove Theorem 5.2 by developing an enumeration algorithm. We provide an example that illustrates important aspects of our algorithm in Appendix C. Fix an OMQ $Q(\bar{x}) = (\mathcal{O}, \mathbf{S}, q_0) \in (\mathbb{G}, \mathrm{CQ})$ with q_0 acyclic and free-connex acyclic, and let an $\mathbf{S}$-database D be given as input.

Preprocessing phase. Recall from Section 3 that the query-directed chase $\mathrm{ch}_{\mathcal{O}}^{q_0}(D)$ can be constructed in time linear in $||D||$. This is the first step of the preprocessing phase. By Lemmas 2.3 and 3.2, we may enumerate $q_0(\mathrm{ch}_{\mathcal{O}}^{q_0}(D))_{\mathbf{N}}^*$ in place of $Q(D)^*$. For brevity, set $D_0 := \mathrm{ch}_{\mathcal{O}}^{q_0}(D)$.

We can assume w.l.o.g. that the tuple $\bar{x}$ has no repeated variables and that q_0 contains no constants and is connected. For connectedness, see Appendix A, for the other properties see the long version. As part of the preprocessing phase, we preprocess q_0 and D_0 in a way that resembles the first phase of the Yannakakis algorithm in which a join tree is traversed in a bottom-up fashion, computing a semi-join in each step [36]. The result is a CQ $q_1(\bar{x})$ and database D_1 that satisfy the following conditions:

(i) q_1 is self-join free, connected (since q_0 is), acyclic, and has no quantified variables (thus is free-connex acyclic); it therefore has a join tree $T_1 = (V_1, E_1)$; we choose a root in T_1 allowing us to speak about predecessors and successors in T_1;
(ii) $\mathrm{adom}(D_1) \subseteq \mathrm{adom}(D_0)$ and for every fact $R(\bar{a}) \in D_1$, there is a fact $S(\bar{b}) \in D_0$ such that $\bar{a}$ and $\bar{b}$ contain exactly the same (database and null) constants;
(iii) $q_0(D_0) = q_1(D_1)$, and thus $q_0(D_0)_{\mathbf{N}}^* = q_1(D_1)_{\mathbf{N}}^*$;
(iv) for all $v = R(\bar{y}) \in V_1$, facts $R(\bar{a}) \in D_1$, and successors $v' = S(\bar{z})$ of v in T_1, D_1 contains a fact $S(\bar{b})$ such that if position i of $\bar{y}$ has the same variable as position j of $\bar{z}$, then position i of $\bar{a}$ has the same constant as position j of $\bar{b}$.

We refer to Condition (iv) as the *progress condition*. Informally, it makes sure that an enumeration algorithm that traverses T_1 in a pre-order tree walk never gets 'stuck' in the sense that it can always extend the partial answer produced so far to a full answer. The construction of q_1 and D_1 is possible in time linear in $||D_0||$. It has been used many times in the context of enumerating answers to conjunctive queries (without ontologies) with constant delay. We give an outline in the long version and refer to [10] for a very clear exposition of the full details. The construction of q_1 and D_1 also tells us whether $q_0(D_0) = \emptyset$. If this is the case, we stop without entering the enumeration phase.

We also use the preprocessing phase to compute data structures that are used in the enumeration phase. We start with some preliminaries. With a *predecessor variable* in an atom $v \in V_1$, we mean a variable that v shares with its predecessor in T_1. By definition, the root of T_1 does not have any predecessor variables. A CQ q is a *subtree* of q_1 if there is a subset $V_q \subseteq V_1$ such that the subgraph $T_q = (V_q, E_1|_{V_q \times V_q})$ of T_1 induced by V_q is connected. Note that q

must be connected since q_1 is and that T_q is a join tree for q. We assume that T_q inherits the direction imposed on T_1 and thus, for instance, may speak about its root.

A *progress tree* is a pair (q, g) with q a subtree of q_1 and $g : \mathsf{var}(q) \to (\mathsf{adom}(D_1) \setminus N) \cup \{*\}$ a map such that the following conditions are satisfied:

(1) $g(x) \neq *$ for every predecessor variable x in the root of T_q;
(2) if $v \in V_q$ and v' is a successor of v in T_1, then $v' \in V_q$ if and only if $g(x) = *$ for some predecessor variable x in v';
(3) there is a homomorphism h from q to D_1 such that for all $x \in \mathsf{var}(q)$, $h(x) \in N$ if $g(x) = *$ and $h(x) = g(x)$ otherwise;
(4) the constants in the range of g form a guarded set in D_1.

To explain the intuition of progress trees, consider a homomorphism h from q_1 to D_1 and an atom $v = R(\bar{y}) \in V_1$ with predecessor variables $\bar{z}$. If $h(\bar{y}) \cap N = \emptyset$, then (v, g) is a (single atom) progress tree, g the restriction of h to the variables in $\bar{y}$. More interesting is the case where $h(\bar{z}) \cap N = \emptyset$, but $h(\bar{y}) \cap N \neq \emptyset$. Informally, under homomorphism h such an atom v 'crosses the boundary' between the 'database part' of D_1 and the 'null part' of D_1. Let $V_q \subseteq V_1$ be the smallest set that contains v and such that if $u \in V_q$ and u' is a successor of u in T_1 such that $h(x) \in N$ for at least one predecessor variable in u', then $u' \in V_q$. This defines a subtree q of q_1 and (q, g) is then a progress tree, where g is the restriction of h to the variables in q with constants from N replaced by $*$. Informally, (q, g) thus describes an 'excursion' of the part q of q_1 into the 'null part' of D_1 and it turns out that properly dealing with such excursions is key to enumerating minimal partial answers. Note that the constants in the range of g form a guarded set in D_1, as required. This relies on q_1 being connected as otherwise, it would be possible to cross the boundary to the null part of D_1 at some guarded set, but return to the database part at a different guarded set.

Consider an atom v in q_1 with predecessor variables $\bar{z}$. A *predecessor map* for v is a function $h : \bar{z} \to \mathsf{adom}(D_1) \setminus N$ that extends to a homomorphism from v to D_1. We call such v and h *relevant*. For all relevant v and h, we compute a linked list $\mathsf{trees}(v, h)$ of all progress trees (q, g) with root v such that $g(\bar{z}) = h(\bar{z})$. We sort the list $\mathsf{trees}(v, h)$ so that it is in *database-preferring order*. This means that progress tree (q, g) is before progress tree (q', g') whenever $(q, g) \prec_{\mathsf{db}} (q', g')$, which is the case if q and q' have the same root and $V_q \subsetneq V_{q'}$, or the following conditions are satisfied for all $x \in \mathsf{var}(q)$:

(a) $V_q = V_{q'}$;
(b) $g(x) = *$ implies $g'(x) = *$;
(c) $g(x) \neq *$ implies $g'(x) \in \{g(x), *\}$;
(d) for some $x \in \mathsf{var}(q)$, $g'(x) = *$ while $g(x) \neq *$.

The algorithm uses these lists as a global data structure that is both accessed and modified. We show in the long version that the lists $\mathsf{trees}(v, h)$ can indeed be computed in linear time on a RAM.

Lemma 5.3. *The lists* $\mathsf{trees}(v, h)$*, for all relevant* v *and* h*, can be computed in overall time linear in* $||D_1||$*. All these lists are non-empty.*

Let $v_0, \ldots, v_k$ be the ordering of the atoms in V_1 generated by a pre-order traversal of T_1. For $v_i \in \{v_0, \ldots, v_k\}$ and a partial map $h : \mathsf{var}(q_1) \to (\mathsf{adom}(D_1) \setminus N) \cup \{*\}$, we use $\mathsf{nextat}_h(v_i)$ to denote v_j with $j > i$ smallest such that $h(x)$ is undefined for some variable x in v_j, if such j exists, and the special symbol eoa (*end of atoms*) otherwise. Clearly, computing nextat is independent of $||D_1||$ and can thus be done in constant time.

Algorithm 1 Enumeration of minimal partial answers.

```
    h_0 = ∅;
    enum(nextat_{h_0}(v_0), h_0);
    function enum(v, h)
        if v = eoa then
5:          output h(x̄);                          % x̄ the variables in q_1
            prune(h);
            return
        let v = R(ȳ) with predecessor variables z̄;
        for all (q, g) ∈ trees(v, h|_z̄) do
10:         h' = h ∪ g;
            enum(nextat_{h'}(v), h');
        return
    function prune(h)
        for all subtrees q of q_1 do
15:         let v be the root of q with predecessor variables z̄;
            for all progress trees (q, g) ≻_db (q, h|_var(q)) do
                remove (q, g) from trees(v, h|_z̄)
        return
```

Enumeration Phase. The enumeration phase of the algorithm is presented in Figure 1. In the **forall** loop in Line 10, we follow the database-preferring order imposed on the trees lists. It is straightforward to show that when a call $\mathsf{enum}(v, h)$ is made, then $v, h|_{\bar{z}}$ used in Line 12 is relevant. The following is an important observation.

Lemma 5.4. *None of the lists* $\mathsf{trees}(v, h)$*, with* v, h *relevant, ever becomes empty.*

Lemma 5.4 is important to achieve constant delay because it implies that that in each call $\mathsf{enum}(v, h)$, the **forall** loop in Line 10 makes at least one iteration and thus at least one recursive call in Line 12. Consequently, while traversing q_1 we never backtrack without producing an output. Note that given v and $h|_{\bar{z}}$, we need to find the (first element of the) list $\mathsf{trees}(v, h|_{\bar{z}})$ in constant time. On a RAM, this can be achieved by a straightforward lookup table.

In the prune subprocedure, there are only constantly many progress trees (q, g) with $(q, g) \succ_{\mathsf{db}} (q, h|_{\mathsf{var}(q)})$ and these can be found in constant time by starting with $g = h|_{\mathsf{var}(q)}$ and then choosing one or more variables $x \in \mathsf{var}(q)$ with $g(x) \neq *$ and setting $g(x) = *$. Note that $(q, h|_{\mathsf{var}(q)})$ is neither required nor guaranteed to be a progress tree. To remove (q', g') from $\mathsf{trees}(v, h|_{\bar{z}})$, we cannot iterate over all progress trees in $\mathsf{trees}(v, h|_{\bar{z}})$ in search of (q', g') as the length of the list is linear. This problem is also solved by a lookup table. When generating the trees lists in the preprocessing phase, we generate a lookup table that takes as argument a progress tree and yields the memory location (register) where that tree is stored as part of a list $\mathsf{trees}(v, h)$. Note that every progress tree occurs in at most one such list. If the list is bidirectionally linked, it is then easy to locate and remove the tree in constant time.

In the long version, we prove the following.

Proposition 5.5. *The algorithm outputs exactly the minimal partial answers to* q_1 *on* D_1*, without repetition.*

6 ENUMERATION WITH MULTI-WILDCARDS

We show that Theorem 5.2 lifts from the case of a single wildcard to the case of multi-wildcards.

Theorem 6.1. *Enumerating minimal partial answers with multi-wildcards is in* $\mathsf{DelayC_{lin}}$ *for OMQs from* $(\mathbb{G}, \mathbb{CQ})$ *that are acyclic and free-connex acyclic.*

Fix an OMQ $Q(\bar{x}) = (\mathcal{O}, \mathbf{S}, q_0) \in (\mathbb{G}, \mathrm{CQ})$ with q_0 acyclic and free-connex acyclic and let an $\mathbf{S}$-database D be given as input. By Lemmas 2.3 and 3.2, we may enumerate $q_0(\mathsf{ch}_{\mathcal{O}}^{q_0}(D))_{\mathbf{N}}^{\mathcal{W}}$ in place of $Q(D)^{\mathcal{W}}$. For brevity, we from now on use D to denote $\mathsf{ch}_{\mathcal{O}}^{q_0}(D)$ (and we will never refer back to the original D).

Our general approach to enumerating $Q^{\mathcal{W}}(D)$ is to combine the enumeration algorithm from Theorem 5.2, here called A_1, with a $\mathsf{DelayC_{lin}}$ algorithm for all-testing (not necessarily minimal) partial answers with multi-wildcards. In fact, we develop such an algorithm A_2 in Appendix B, which is non-trivial. With the algorithm in place, a first (incomplete) implementation of the general approach could then be as follows. Use A_1 to enumerate $Q^*(D)$. For each obtained answer $\bar{a}^*$, construct the *multi-wildcard ball of* $\bar{a}^*$, that is, the set $B^{\mathcal{W}}(\bar{a}^*)$ of multi-wildcard tuples $\bar{a}^{\mathcal{W}}$ such that replacing all occurrences of wildcards from $\mathcal{W}$ in $\bar{a}^{\mathcal{W}}$ by the single-wildcard '$*$' results in $\bar{a}^*$. Note that as the length of $\bar{a}^*$ is bounded by a constant, so is the cardinality of $B^{\mathcal{W}}(\bar{a}^*)$. Discard from $B^{\mathcal{W}}(\bar{a}^*)$ those tuples that are not partial answers using A_2, and then output those among the remaining tuples that are minimal w.r.t. '$\prec$'.

Example 6.2. Let $Q = (\mathcal{O}, \mathbf{S}, q_0)$ where

$$\mathcal{O} = \{A(x) \rightarrow \exists y_1 \exists y_2\, R(x, y_1) \wedge T(x, y_1) \wedge S(x, y_2)\},$$

$\mathbf{S}$ contains all relation symbols in $\mathcal{Q}$, and

$$q_0(x_0, x_1, x_2, x_3) = R(x_0, x_1) \wedge S(x_0, x_2) \wedge T(x_0, x_3).$$

Further let $D = \{A(c), R(c, c')\}$. Then $Q^*(D) = \{(c, c', *, *)\}$ and $Q^{\mathcal{W}}(D) = \{(c, c', *_1, *_2), (c, *_1, *_2, *_1)\}$. But we never consider (and thus do not output) the multi-wildcard tuple $(c, *_1, *_2, *_1)$.

The solution involves replacing the multi-wildcard ball $B^{\mathcal{W}}(\bar{a}^*)$ with the *multi-wildcard cone*

$$\mathsf{cone}^{\mathcal{W}}(\bar{a}^*) = \bigcup_{\bar{b}^* : \bar{a}^* \preceq \bar{b}^*} B^{\mathcal{W}}(\bar{b}^*).$$

Clearly, also the cardinality of $\mathsf{cone}^{\mathcal{W}}(\bar{a}^*)$ is bounded by a constant. Regarding Example 6.2, note that $(c, *_1, *_2, *_1) \notin B^{\mathcal{W}}(c, c', *, *)$, but $(c, *_1, *_2, *_1) \in \mathsf{cone}^{\mathcal{W}}(c, c', *, *)$. However, the cones of different tuples $\bar{a}^*, \bar{b}^* \in Q^*(D)$ might overlap and thus for some $\bar{a}^* \in Q^*(D)$, there might be no tuple in $\mathsf{cone}^{\mathcal{W}}(\bar{a}^*)$ that we haven't yet output, compromising constant delay. We address these issues by using a careful combination of balls, cones, and pruning.

The preprocessing phase consists of running the preprocessing phases of A_1 and A_2. The enumeration phase is shown in Figure 2. With L, we denote a bidirectionally linked list in which we store multi-wildcard tuples and that is initialized as the empty list. In the **forall** loop in Line 2, we use algorithm A_1 to iterate over all minimal partial answers in $q(D)_{\mathbf{N}}^*$. With $q(D)_{\mathbf{N}}^{\mathcal{W},\not\prec}$, we denote the set of (not necessarily minimal) partial answers with multi-wildcards to CQ q on database D. The intersections with $q(D)_{\mathbf{N}}^{\mathcal{W},\not\prec}$ in Line 3 and 7 can

Algorithm 2 Enumeration of minimal partial answers with multi-wildcards.

```
    L = [];
    for all ā* ∈ q(D)*_N do
3:      for all ā^W ∈ cone^W(ā*) ∩ q(D)^{W,⊀}_N with F(ā^W) = 0 do
            F(ā^W) = 1;
            append ā^W to L;
6:          prune(ā^W)
        choose ā^W ∈ min^≺(B^W(ā*) ∩ q(D)^{W,⊀}_N);
        output ā^W;
9:      remove ā^W from L;
    output all tuples in L;
    return
12: function prune(ā^W)
        for all multi-wildcard tuples b̄^W such that ā^W ≺ b̄^W do
            F(b̄^W) = 1;
15:         remove b̄^W from L;
        return
```

be computed in constant time using algorithm A_2. F is a lookup table that stores a Boolean value for every multi-wildcard tuple of length $|\bar{x}|$, initialized with 0 (all memory is initialized with 0 in our machine model). Informally, $F(\bar{a}^{\mathcal{W}})$ is set to 1 if $\bar{a}^{\mathcal{W}}$ has already been added to the list L or is not in $q(D)_{\mathbf{N}}^{\mathcal{W}}$ (and thus does not need to be added to L). For a set of multi-wildcard tuples S, we use $\min^{\prec}(S)$ to denote the tuples in S that are minimal w.r.t. '$\prec$'. To remove multi-wildcard tuples from L in constant time, we use a lookup table that stores, for every multi-wildcard tuple $\bar{a}^{\mathcal{W}}$ on the list L, the memory location of the list node representing $\bar{a}^{\mathcal{W}}$. Since L is bidirectionally linked, this allows us to remove $\bar{a}^{\mathcal{W}}$ from L in constant time. Since the arity of relation symbols is (implicitly) bounded by a constant, so is the number of iterations of the **forall** loop in Line 13. In summary, the preprocessing phase runs in linear time while the enumeration phase has only constant delay. In the long version we prove the following.

Lemma 6.3. *The algorithm outputs exactly the minimal partial answers with multi-wildcards to q on D, without repetition.*

7 CONCLUSIONS

As future work, it would be interesting to consider as the ontology language also description logics with functional roles such as $\mathcal{ELIF}$; there should be a close connection to enumeration of answers to CQs in the presence of functional dependencies [20]. A much more daring extension would be to $(\mathbb{G}, \mathbb{UCQ})$ or even to $(\mathbb{FG}, (\mathbb{U})\mathbb{CQ})$ where $\mathbb{UCQ}$ denotes unions of CQs and $\mathbb{FG}$ denotes frontier-guarded TGDs. Note, however, that enumeration in CD∘Lin of answers to UCQs is not fully understood even in the case without ontologies [21]. Another interesting question is whether the enumeration problems placed in $\mathsf{DelayC_{lin}}$ in the current paper actually fall within CD∘Lin, that is, whether the use of a polynomial amount of memory in the enumeration phase can be avoided.

Acknowledgement. We acknowledge support by the DFG project LU 1417/3-1 'QTEC'.

REFERENCES

[1] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of FOCS 2014*, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: http://webdam.inria.fr/Alice/.

[3] Franz Baader, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017. doi:10.1017/9781139025355.

[4] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Proceedings of CSL 2007*, volume 4646, pages 208–222, 2007. doi:10.1007/978-3-540-74915-8_18.

[5] Jean-François Baget, Marie-Laure Mugnier, Sebastian Rudolph, and Michaël Thomazo. Walking the complexity lines for generalized guarded existential rules. In *Proceedings of IJCAI 2011*, pages 712–717. IJCAI/AAAI, 2011. doi:10.5591/978-1-57735-516-8/IJCAI11-126.

[6] Pablo Barceló, Victor Dalmau, Cristina Feier, Carsten Lutz, and Andreas Pieris. The limits of efficiency for open- and closed-world query evaluation under guarded TGDs. In *Proceedings of PODS 2020*, pages 259–270, 2020. doi:10.1145/3375395.3387653.

[7] Pablo Barceló, Cristina Feier, Carsten Lutz, and Andreas Pieris. When is ontology-mediated querying efficient? In *Proceedings of LICS 2019*, pages 1–13, 2019. doi:10.1109/LICS.2019.8785823.

[8] Pablo Barceló, Diego Figueira, Georg Gottlob, and Andreas Pieris. Semantic optimization of conjunctive queries. *J. ACM*, 67(6):34:1–34:60, 2020. doi:10.1145/3424908.

[9] Pablo Barceló, Reinhard Pichler, and Sebastian Skritek. Efficient evaluation and approximation of well-designed pattern trees. In *Proceedings of PODS 2015*, pages 131–144. ACM, 2015. doi:10.1145/2745754.2745767.

[10] Christoph Berkholz, Fabian Gerhardt, and Nicole Schweikardt. Constant delay enumeration for conjunctive queries: a tutorial. *ACM SIGLOG News*, 7(1):4–33, 2020. doi:10.1145/3385634.3385636.

[11] Christoph Berkholz and Nicole Schweikardt. Constant delay enumeration with fpt-preprocessing for conjunctive queries of bounded submodular width. In *Proceedings of MFCS 2019*, pages 58:1–58:15, 2019. doi:10.4230/LIPIcs.MFCS.2019.58.

[12] Meghyn Bienvenu and Magdalena Ortiz. Ontology-mediated query answering with data-tractable description logics. In *Proceedings of Reasoning Web*, pages 218–307, 2015. doi:10.1007/978-3-319-21768-0_9.

[13] Meghyn Bienvenu, Magdalena Ortiz, Mantas Simkus, and Guohui Xiao. Tractable queries for lightweight description logics. In *Proceedings of IJCAI 2013*, pages 768–774. IJCAI/AAAI, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6908.

[14] Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive datalog, CSP, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44, 2014. doi:10.1145/2661643.

[15] Endre Boros, Benny Kimelfeld, Reinhard Pichler, and Nicole Schweikardt. Enumeration in data management (Dagstuhl seminar 19211). *Dagstuhl Reports*, 9(5):89–109, 2019. doi:10.4230/DagRep.9.5.89.

[16] Johann Brault-Baron. *De la pertinence de l'énumération : complexité en logiques propositionnelle et du premier ordre. (On the relevance of enumeration: complexity of propositional and first-order logic)*. PhD thesis, University of Caen Normandy, France, 2013. URL: https://tel.archives-ouvertes.fr/tel-01081392.

[17] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013. doi:10.1613/jair.3873.

[18] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *J. Web Semant.*, 14:57–83, 2012. doi:10.1016/j.websem.2012.03.001.

[19] Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012. doi:10.1016/j.artint.2012.08.002.

[20] Nofar Carmeli and Markus Kröll. Enumeration complexity of conjunctive queries with functional dependencies. *Theory Comput. Syst.*, 64(5):828–860, 2020. doi:10.1007/s00224-019-09937-9.

[21] Nofar Carmeli and Markus Kröll. On the enumeration complexity of unions of conjunctive queries. *ACM Trans. Database Syst.*, 46(2):5:1–5:41, 2021. doi:10.1145/3450263.

[22] Nofar Carmeli, Shai Zeevi, Christoph Berkholz, Benny Kimelfeld, and Nicole Schweikardt. Answering (unions of) conjunctive queries using random access and random-order enumeration. In *Proceedings of PODS 2020*, pages 393–409, 2020. doi:10.1145/3375395.3387662.

[23] Stephen A. Cook and Robert A. Reckhow. Time bounded random access machines. *J. Comput. Syst. Sci.*, 7(4):354–375, 1973. doi:10.1016/S0022-0000(73)80029-7.

[24] Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *Proceedings of Principles and Practice of Constraint Programming - CP 2002*, pages 310–326, 2002. doi:10.1007/3-540-46135-3_21.

[25] Shaleen Deep, Xiao Hu, and Paraschos Koutris. Enumeration algorithms for conjunctive queries with projection. In *Proceedings of ICDT 2021*, pages 14:1–14:17, 2021. doi:10.4230/LIPIcs.ICDT.2021.14.

[26] Shaleen Deep and Paraschos Koutris. Ranked enumeration of conjunctive query results. In *Proceedings of ICDT 2021*, pages 5:1–5:19, 2021. doi:10.4230/LIPIcs.ICDT.2021.5.

[27] William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *The Journal of Logic Programming*, 1(3):267–284, 1984. doi:10.1016/0743-1066(84)90014-1.

[28] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *J. Theor. Comput. Sci.*, 336(1):89–124, 2005. doi:10.1016/j.tcs.2004.10.033.

[29] Etienne Grandjean. Sorting, linear time and the satisfiability problem. *Annals of Mathematics and Artificial Intelligence*, 16:183–236, 1996. doi:10.1007/BF02127798.

[30] David S. Johnson and Anthony C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984. doi:10.1016/0022-0000(84)90081-3.

[31] Wojciech Kazana. *Query evaluation with constant delay. (L'évaluation de requêtes avec un délai constant)*. PhD thesis, École normale supérieure de Cachan, Paris, France, 2013. URL: https://tel.archives-ouvertes.fr/tel-00919786.

[32] Markus Kröll, Reinhard Pichler, and Sebastian Skritek. On the complexity of enumerating the answers to well-designed pattern trees. In *Proceedings of ICDT 2016*, pages 22:1–22:18, 2016. doi:10.4230/LIPIcs.ICDT.2016.22.

[33] Carsten Lutz and Marcin Przybyłko. Efficiently enumerating answers to ontology-mediated queries. *CoRR*, abs/2203.09288, 2022. doi:10.48550/arXiv.2203.09288.

[34] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, pages 455–469, 1979. doi:10.1145/320107.320115.

[35] Luc Segoufin. Constant delay enumeration for conjunctive queries. *SIGMOD Rec.*, 44(1):10–17, 2015. doi:10.1145/2783888.2783894.

[36] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the Seventh International Conference on Very Large Data Bases - Volume 7*, pages 82–94, 1981.

This is the appendix of the conference version of the paper, which is subject to space restrictions. Full proofs can be found in [33].

A CONNECTED QUERIES IN SECTION 5

We argue that, when enumerating minimal partial answers with a single wildcard, we can indeed assume the query to be connected. For assume that we have found an enumeration algorithm for connected CQs that runs in $\mathsf{DelayC_{lin}}$. We can then enumerate $q_0(D_0)^*_{\mathbf{N}}$ in $\mathsf{DelayC_{lin}}$ when q_0 has maximal connected components $p_0, \ldots, p_k$, $k > 0$, in the following way. We first do preprocessing for all $p_0, \ldots, p_k$. We then start an algorithm that enumerates the answers to p_0. After the first answer was found, it calls the enumeration algorithm for p_1, which upon finding an answer calls the enumeration algorithm for p_2, and so on. Only when the innermost algorithm found an answer to p_k, the answers are combined and output as an answer to Q. Note the algorithms for $p_1, \ldots, p_k$ have to start from scratch multiple times which is problematic since the data structures computed in the preprocessing phase are modified in the enumeration phase and we cannot repeat preprocessing because that would introduce a linear time delay into the enumeration phase. An easy solution is as follows. When first enumerating the answers to p_k, we store all of them in the form of a linked list. When we need to enumerate the answers to p_k again, we can just use that list without any preprocessing. We do the same for the subqueries $p_{k-1} \cup p_k, p_{k-2} \cup p_{k-1} \cup p_k$, and so on, which fixes the problem. The above argument requires a polynomial amount of memory during the enumeration phase. There is, however, an alternative approach that avoids this. Our algorithm is such that the data structure S computed in the preprocessing phase is modified in the enumeration phase, resulting in a data structure S'. However, S' is such that it could have been used in place of S after the preprocessing phase without affecting the output of enumeration. This means that the preprocessing can simply be skipped before restarting the enumeration algorithm for a connected subquery.

B ALL-TESTING PARTIAL ANSWERS WITH MULTI-WILDCARDS

The algorithm for enumerating minimal partial answers with multi-wildcards presented in Section 6 relies on a $\mathsf{DelayC_{lin}}$ algorithm for all-testing (not necessarily minimal) partial answers with multi-wildcards. In this section, we develop such an algorithm.

Proposition B.1. *For every CQ $q(\bar{x})$ that is acyclic and free-connex acyclic, all-testing of the answers $q(D)^{\mathcal{W},\not<}_{\mathbf{N}}$ is in $\mathsf{DelayC_{lin}}$ for databases D of the form $\mathsf{ch}^q_O(D_0)$ and sets of nulls $N = \mathsf{adom}(D) \setminus \mathsf{adom}(D_0)$.*

To prove Proposition B.1, fix a CQ $q(\bar{x})$ over schema S that is acyclic and free-connex acyclic, and let D be an S-database and N a set of nulls satisfying the conditions from Proposition B.1. In time linear in $||D||$ we can convert q and D into a CQ $q'(\bar{x})$ without quantified variables and a database D' such that D and D' have the same Gaifman graph and $q(D) = q'(D')$, and thus also $q(D)^{\mathcal{W},\not<}_{\mathbf{N}} = q'(D')^{\mathcal{W},\not<}_{\mathbf{N}}$. Note that we achieve this as part of the preprocessing carried out in Section 5. We may substitute q with q' and D with D', in effect simply assuming that q contains no quantified variables.

Let $T_q = (V_q, E_q)$ be a join tree for q. A *multi-progress tree* is a pair (q', g) with q' a subtree of q (as defined in Section 5) and $g : \mathsf{var}(q) \to (\mathsf{adom}(D) \setminus N) \cup \mathcal{W}$ a map such that the following conditions are satisfied:

(1) $g(x) \notin \mathcal{W}$ for every predecessor variable x in the root of $T_{q'}$;
(2) if $v \in V_{q'}$ and v' is a successor of v in T_q, then $v' \in V_{q'}$ if and only if $g(x) \in \mathcal{W}$ for some predecessor variable x in v';
(3) the constants in the range of g form a guarded set in D.

Let $D_1, \ldots, D_n$ be the tree-like structures in D generated by the chase. A set $S = \{(q_1, g_1), \ldots, (q_\ell, g_\ell)\}$ of multi-progress trees is *valid* if there is a homomorphism h from $q_1 \cup \cdots \cup q_\ell$ to some database D_i, with $1 \leq i \leq n$, that is *compatible* with g, that is, for all $x, y \in \mathsf{var}(q_1) \cup \cdots \cup \mathsf{var}(q_\ell)$,

(a) $h(x) \in N$ if $g(x) \in \mathcal{W}$ and $h(x) = g(x)$ otherwise;
(b) $g(x) = g(y)$ implies $h(x) = h(y)$.

Our $\mathsf{DelayC_{lin}}$ algorithm for all-testing $q(D)^{\mathcal{W},\not<}_{\mathbf{N}}$ uses as a black box a $\mathsf{DelayC_{lin}}$ algorithm $A_{q'}$ for all-testing of $q'(D)$, for every subquery $q'(\bar{x}')$ of $q(\bar{x})$, that is, for every CQ $q'(\bar{x}')$ that can be obtained from q by dropping atoms. Note that all of these q' contain no quantified variables and are thus free-connex acyclic, implying that all-testing $q'(D)$ is possible in $\mathsf{DelayC_{lin}}$ by Proposition 4.2. There are clearly only constantly many such subqueries.

In the preprocessing phase, we run the preprocessing phases of all the algorithms $A_{q'}$, q' a subquery of q. In addition, we precompute a lookup table $\mathsf{nullhom}$ that stores a Boolean value for all sets S of multi-progress trees that contain at most $|\mathsf{var}(q)|$ such trees. Let $S = \{(q_1, g_1), \ldots, (q_\ell, g_\ell)\}$. The stored value is 1 if S is valid and 0 otherwise. Such a lookup table can be accessed and updated in $O(1)$ time on a RAM. The proof of the following is similar to that of Lemma 5.3.

Lemma B.2. *The lookup table* $\mathsf{nullhom}$ *can be computed in time linear in $||D||$.*

We now describe the testing phase of our algorithm. Assume that a multi-wildcard tuple $\bar{a}^{\curlywedge\mathcal{W}}$ of length $|\bar{x}|$ is to be tested. We may first check whether wildcards are used in the required way and answer 'no' if this is not the case. More precisely, we check that the wildcards in $\bar{a}^{\curlywedge\mathcal{W}}$ are a prefix of the ordered set $\mathcal{W} = \{*_1, *_2, \ldots\}$ and that multiple occurrences of the same variable in $\bar{x}$ are matched by multiple occurrences of the same wildcard in $\bar{a}^{\curlywedge\mathcal{W}}$. If this is the case, we may view $\bar{a}^{\curlywedge\mathcal{W}}$ as a map $h_{\bar{a}^{\curlywedge\mathcal{W}}} : \mathsf{var}(q) \to (\mathsf{adom}(D) \setminus N) \cup \mathcal{W}$ in the obvious way. We may then check that the wildcards in $\bar{a}^{\curlywedge\mathcal{W}}$ respect the order of the answer variables in $\bar{x}$, that is, if the first occurrence of x is before the first occurrence of x' in $\bar{x}$, $h_{\bar{a}^{\curlywedge\mathcal{W}}}(x) = *_i$, and $h_{\bar{a}^{\curlywedge\mathcal{W}}}(x') = *_j$, then $i < j$.

We say that a multi-progress tree (q', g) is *realized* in $\bar{a}^{\curlywedge\mathcal{W}}$ if $h_{\bar{a}^{\curlywedge\mathcal{W}}}(x) = g(x)$ for all $x \in \mathsf{var}(q')$. Let T be the set of all multi-progress trees realized in $\bar{a}^{\curlywedge\mathcal{W}}$ and let $\sim$ be the smallest equivalence relation on T such that $(q_1, g_1) \sim (q_2, g_2)$ if there are variables $x_1 \in \mathsf{var}(q_1)$ and $x_2 \in \mathsf{var}(q_2)$ such that $g_1(x_1) = g_2(x_2) \in \mathcal{W}$. We consider each equivalence class $S \subseteq T$ of '$\sim$' and check whether S is valid by testing if $\mathsf{nullhom}(S) = 1$. If any of the checks fails, we answer 'no'. Since at most $|\mathsf{var}(q)|$ (and thus only constantly many) multi-progress trees may be realized in $\bar{a}^{\curlywedge\mathcal{W}}$, the required checks can be done in constant time.

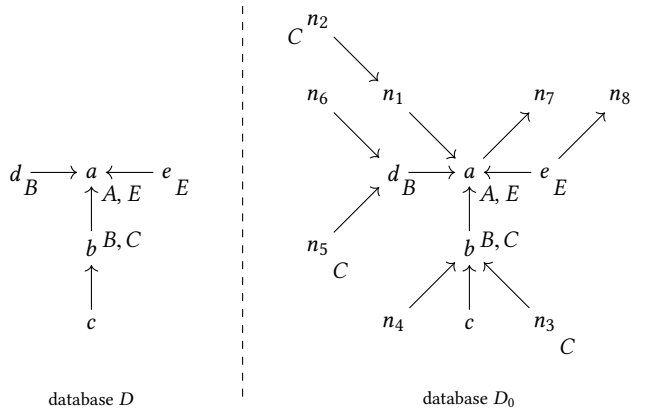

Figure 3: Database D and query-directed chase D_0. All edges represent relation R and every constant has an L-self-loop that is not shown.

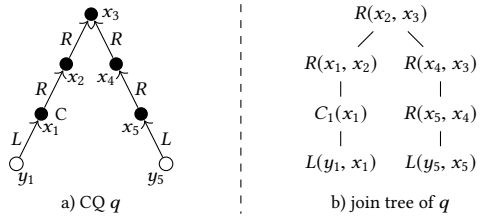

Figure 2: CQ q and its join tree.

We then do one last check. Let q' be the subquery of q that consists of all atoms α such that for all variables x in α, $h_{\bar{a}'\mathcal{W}}(x) \notin \mathcal{W}$. Further let $\bar{a}$ be the tuple over $\mathrm{adom}(D) \setminus N$ obtained from $\bar{a}'^{\mathcal{W}} = (a_1'^{\mathcal{W}}, \ldots, a_{|\bar{x}|}'^{\mathcal{W}})$ by dropping $a_i'^{\mathcal{W}}$ whenever the i-th position in $\bar{x}$ is an answer variable that is not in $\mathrm{var}(q')$. We then use algorithm $A_{q'}$ to test whether $\bar{a} \in q'(D)$ and return the result. The following lemma asserts that the returned answer is correct, which finishes the proof of Proposition 4.2.

Lemma B.3. *$\bar{a}'^{\mathcal{W}} \in q(D)^{\mathcal{W}}_{\mathrm{N}}$ iff the testing phase returns 'yes'.*

C ILLUSTRATING THE ALGORITHM

We give examples that showcase important aspects of the enumeration algorithm for minimal partial answers with a single wildcard presented in Section 5.

Assume that the enumeration algorithm is started on the OMQ $Q(\bar{x}) = (\mathcal{O}, \mathbf{S}, q) \in (\mathbb{G}, \mathbb{CQ})$ where $\mathcal{O}$ consists of the TGDs

$$\begin{aligned} A(x) &\rightarrow \exists y_1 \exists y_2\ R(y_1, y_2) \wedge R(y_2, x) \wedge C(y_2) \\ B(x) &\rightarrow \exists y_1 \exists y_2\ R(y_1, x) \wedge R(y_2, x) \wedge C(y_1) \\ E(x) &\rightarrow \exists y_1\ R(x, y_1) \\ R(x, y) &\rightarrow L(x, x) \wedge L(y, y), \end{aligned}$$

the schema $\mathbf{S}$ is $\{A, B, C, E, R\}$, and where q is the CQ

$$\begin{aligned} q(\bar{x}) \leftarrow\ & \exists y_1 \exists y_5\ L(y_1, x_1), R(x_1, x_2), R(x_2, x_3), \\ & R(x_4, x_3), R(x_5, x_4), L(y_5, x_5), C(x_1). \end{aligned}$$

with $\bar{x} = (x_1, x_2, x_3, x_4, x_5)$. The CQ q is displayed in Figure 2. It is acyclic and free-connex acyclic, as witnessed by the join trees for q and its extension $\hat{q}$ with the atom $\hat{R}(x_1, \ldots, x_5)$. The join tree for q is given in Figure 2. Note that the atoms that contain only answer variables constitute a connected prefix of the join tree of q. This can (almost[1]) always be achieved for CQs that are acyclic and free-connex acyclic and is exploited in the preprocessing phase.

Assume that the input database D is as depicted on the left-hand side of Figure 3, where all edges represent the relation symbol R.

Preprocessing. In the preprocessing phase, we modify the query q and database D to obtain the CQ q_2 and database D_2 that are used in the enumeration phase. This is done in several steps. In the very first step, we set $q_0 = q$ and replace D with the query-directed chase $D_0 = \mathrm{ch}^q_{\mathcal{O}}(D)$, displayed on the right-hand side of Figure 3.

The next step is to construct from q_0 and D_0 a self-join free CQ q_1 without quantified and a database D_1 that has been adjusted accordingly. It is this step that exploits the special shape of the join tree of q_0 mentioned above. In our case, q_1 is

$$q_1(\bar{x}) \leftarrow R_1(x_1, x_2), R_2(x_2, x_3), R_4(x_4, x_3), R_5(x_5, x_4), C_1(x_1).$$

Informally, q_1 was obtained from q by renaming relation symbols to achieve self-join freeness and dropping atoms that involve a quantified variable. The join tree of q_1 is the join tree of q except that relation symbols in atoms change and nodes/atoms that contain any of the variables y_1, y_5 are removed.

The database D_1 is shown in Figure 3 where, for better readability, we only show the index i of edge labels R_i. Observe that the constant n_8 was removed and that edges are now multi-edges. To get an intuition of the construction of D_1, consider the fact $R(n_2, n_1)$ in D_0. In principle, any of the four R-atoms in q_0 can map to it, and in q_1 the relation symbol R in those atoms has been renamed to R_1, R_2, R_4, and R_5, respectively. Thus, we should be prepared to include in D_1 the fact $R_i(n_2, n_1)$ for all $i \in \{1, 2, 4, 5\}$. However, a closer inspection shows that the atom $R(x_2, x_3)$ in q_0 cannot map to the fact $R(n_2, n_1)$ in D_0 since then x_1 would have to be mapped to an R-predecessor of n_2, which does not exist. A similar observation holds for the atom $R(x_4, x_3)$ in q_0 and thus we only include in D_1 the facts $R_1(n_2, n_1)$ and $R_5(n_2, n_1)$. The 'right' facts to include are identified during a bottom-up walk over the join-tree of q_0. Note that the relation symbols A, B, have been dropped since they do not occur in q_0.

Lists of progress trees. The last step of the preprocessing phase is to create the lists $\mathrm{trees}(v, h)$ of progress trees for each atom v in q_2 and each predecessor map h for v. Recall that by the latter we mean a function $h : \bar{z} \rightarrow \mathrm{adom}(D_2) \setminus N$ whose range is a guarded set in D_2, and where $\bar{z}$ are the predecessor variables in v. For brevity, we represent h in the form $z_1 \cdots z_n \mapsto c_1 \cdots c_n$ when $z_1, \ldots, z_n$ are the variables in $\bar{z}$ and $h(z_i) = c_i$ for $1 \leq i \leq n$; this becomes $\varepsilon \mapsto \varepsilon$ when $\bar{z}$ is the empty tuple. For the join tree of q_1 with marked predecessor variables, see Figure 5.

Also recall that a progress tree is a pair (q, g) with CQ q a subtree of q_2 and g a function from $\mathrm{var}(q)$ to $(\mathrm{adom}(D_2) \setminus N) \cup \{*\}$ that must satisfy Conditions (1)-(4) given in Section 5. We represent

[1] It can always be achieved when using a generalized hypertree decomposition of width 1 in place of a join tree, see [10].

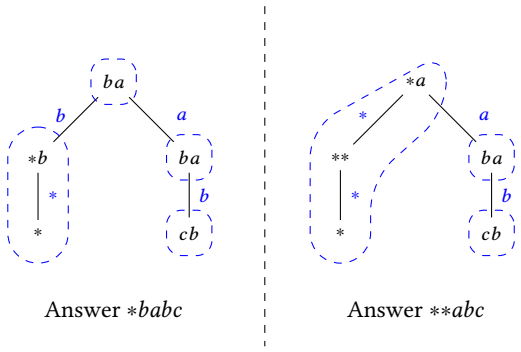

Figure 6: Three least partial answers, inside join tree of q_2.

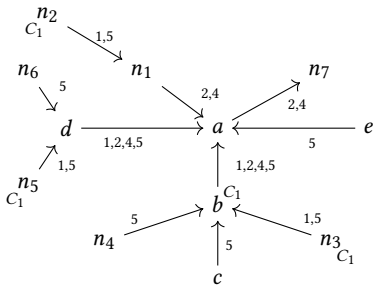

Figure 4: Database D_1. Edges are labeled with indices of the relation symbols R_1, R_2, R_4, R_5 that constitute the edge.

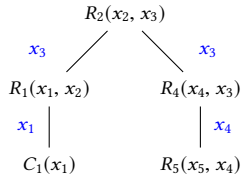

Figure 5: Join tree of q_2. The predecessor variables of each atom are shown on the incoming edge of the atom.

the function g in the same way as predecessor maps. Examples of progress trees include

$$(R_1(x_1, x_2), x_1x_1x_2 \mapsto ba)$$

and

$$(R_2(x_2, x_3) \wedge R_1(x_1, x_2) \wedge C_1(x_1), x_1x_2x_3 \mapsto **a).$$

The reader is invited to verify that the relevant Conditions (1)-(4) are all satisfied for these progress trees. Intuitively, the second progress tree (q, g) above describes an 'excursion' of the part q of q_2 into the 'null part' of D_2. This excursion consists of mapping x_1 to n_2, x_2 to n_1, and x_3 to a.

Let us review two non-examples progress trees, starting with

$$(R_4(x_4, x_3) \wedge R_5(x_5, x_4), x_3x_4x_5 \mapsto abc)$$

which is not a progress tree as the predecessor variable x_4 of atom $R_5(x_5, x_4)$ is mapped to '$*$' and thus Condition (2) is violated. Next consider

$$(R_4(x_4, x_3) \wedge R_5(x_5, x_4), x_3x_4x_5 \mapsto a*c)$$

which is not a progress tree because there is no guarded set in D_2 that contains a and c, and thus Condition (4) is violated.

We now give all the lists trees(v, h) that are computed in the preprocessing phase. For brevity, we represent progress trees (q, g) as the CQ q in which every variable x was replaced with $g(x)$. List items are separated by ';'. The lists are:

- atom $v = R_1(x_1, x_2)$ with a predecessor variable x_2
  - trees$(v, x_2 \mapsto a) = [R_1(b, a); R_1(d, a)]$
  - trees$(v, x_2 \mapsto b) = [R_1(*, b) \wedge C_1(*)]$
  - trees$(v, x_2 \mapsto c) = []$
  - trees$(v, x_2 \mapsto d) = [R_1(*, d) \wedge C_1(*)]$
  - trees$(v, x_2 \mapsto e) = []$
- atom $v = R_2(x_2, x_3)$ no predecessor variables
  - trees$(v, \emptyset \mapsto \emptyset) = [R_2(b, a); R_2(d, a);$ $R_2(*, a) \wedge R_1(*, *) \wedge C_1(*); R_2(a, *) \wedge R_4(a, *)]$
- atom $v = R_4(x_4, x_3)$ with a predecessor variable x_3
  - trees$(v, x_3 \mapsto \beta) = []$ for $\beta \in \{b, c, d, e\}$
  - trees$(v, x_3 \mapsto c_0a) = [R_4(b, a); R_4(d, a);$ $R_4(*, a) \wedge R_5(*, *)]$
- atom $v = R_5(x_5, x_4)$ with a predecessor variable x_4
  - trees$(v, x_2 \mapsto c) = []$
  - trees$(v, x_2 \mapsto a) = [R_5(b, a); R_5(d, a); R_5(e, a)]$
  - trees$(v, x_2 \mapsto b) = [R_5(c, b); R_5(*, b)]$
  - trees$(v, x_2 \mapsto d) = [R_5(*, d)]$
  - trees$(v, x_2 \mapsto e) = []$
- atom $v = C_1(x_1)$ with a predecessor variable x_1
  - trees$(v, x_1 \mapsto \beta) = []$ for $\beta \in \{a, c, d, e\}$
  - trees$(v, x_1 \mapsto b) = [C_1(b)]$

All the remaining lists are empty. The lists above are sorted in database preferring order, as required, and thus we are ready for the enumeration phase.

Enumeration and pruning. In the enumeration phase, we traverse the join tree of q_1 in a depth-first fashion, assembling a minimal partial answer to q_1 on D_1. Once such an answer is found, we output it and execute pruning, then backtrack in a systematic way and re-start answer assemblage to produce the next answer, and so on.

In our example, there are no complete answers. The first partial answer generated is $\bar{c}^* = *babc$. The answer $\bar{c}^*$ is displayed on the left-hand side of Figure 6, inside the join tree for q_1. The blue boxes indicate the progress trees that have been used in assembling the answer $\bar{c}^*$.

Let us now consider pruning with $\bar{c}^*$. Informally, we consider all progress trees (q, g) such that q is some subtree of q_2 and g can be obtained by starting with $\bar{x} \mapsto \bar{c}^*$, then restricting to the variables in var(q), and then switching at least one variable from a non-wildcard to a wildcard. One example of such a progress tree is

$$R_2(*, a) \wedge R_1(*, *) \wedge C_1(*).$$

Pruning removes this tree from trees$(R_2(x_2, x_3), \emptyset \mapsto \emptyset)$. One consequence of this pruning that the partial answer $**abc$ displayed on the right of Figure 6, which is not a minimal partial answer, is not output in the enumeration phase.