

Computing ABox Justifications for Query Answers via Datalog Rewriting

Stefan Borgwardt, Steffen Breuer and Alisa Kovtunova

Institute of Theoretical Computer Science, Technische Universität Dresden, 01062 Dresden, Germany

Abstract


Justifications are a very useful tool for explaining DL consequences. They highlight parts of the ontology that are responsible for the consequence, and can serve as the basis for more detailed explanations such as proofs. In this paper, we present an approach that can compute ABox justifications, i.e., justifications restricted to assertions, for answers to conjunctive queries, assuming that these queries are Datalog-rewritable over the input ontology. We implemented the approach based on the rewriting tool Clipper and ProvSQL, which can be used to compute provenance information, including justifications, for SQL queries. The potentially recursive nature of Datalog rewritings does not allow a direct translation into SQL queries, but requires some additional processing steps, depending on the cyclic structure of the Datalog program and the ABox. We show that the set of all ABox justifications can be computed in reasonable time, and compare the performance with Soufflé, a Datalog engine that also supports explanations.


Keywords


Query Answering, Justifications, Explanation, Provenance


1. Introduction


Despite the reputation of description logics and other logic-based formalisms of supporting the inspection and understanding of large and complex domain knowledge, sometimes DL entailments are far from understandable by themselves. For this reason, research on explanation approaches for DLs already has a long history. *Justifications*, which point out the responsible axioms, are the most popular tool and often enough to understand a given entailment [1, 2, 3, 4], and black-box justification techniques have long been supported in the ontology editor Protégé [5].¹ More detailed *proofs* of entailments have been investigated for a longer time, but received less attention so far [6, 7, 8, 9, 10]. They are especially helpful when the justifications are large or require unintuitive reasoning steps to reach the conclusion. However, justifications are also useful for computing proofs since they can be used as a preprocessing step to filter irrelevant axioms from the ontology and make proof computation more efficient [9, 11].


 DL 2023: 36th International Workshop on Description Logics, September 2–4, 2023, Rhodes, Greece

 stefan.borgwardt@tu-dresden.de (S. Borgwardt); steffen.breuer@mailbox.tu-dresden.de (S. Breuer); alisa.kovtunova@tu-dresden.de (A. Kovtunova)

 <https://lat.inf.tu-dresden.de/~stefborg/> (S. Borgwardt); <https://lat.inf.tu-dresden.de/~alisa/> (A. Kovtunova)

 0000-0003-0924-8478 (S. Borgwardt); 0000-0001-9936-0943 (A. Kovtunova)

 © 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

¹<https://github.com/protegeproject/explanation-workbench>

Related areas of research include *provenance* for databases [12, 13, 14], which is more general than justifications, but does not consider ontologies. Instead of arbitrary provenance semirings [15], justifications correspond to the so-called *why-provenance* [16]. There are also provenance formalisms that can deal with the recursive nature of Datalog programs [17, 18, 19], but they are not necessary to compute the why-provenance. Provenance has been considered also for *DL-Lite* and \mathcal{EL} ontologies, taking the problem of recursive derivations into account by, e.g. restricting to idempotent semirings [20, 21]. Less closely related is work on proofs for query answering in DLs or Datalog [22, 19, 23].

In this paper, we consider a type of justifications for *conjunctive query answering* in Datalog-rewritable description logics. We report on an implementation that combines the Datalog rewritings computed by Clipper, which supports Horn-*SHIQ* ontologies [24], with ProvSQL, a tool for computing provenance for database queries [14]. In case the Datalog rewriting is non-recursive, it can be directly translated into SQL queries over the database, which are then processed by ProvSQL. For recursive Datalog programs, we need to do some additional processing based on the cyclic structures in the program and the data. For each answer to the original query, this method computes all *ABox justifications*, i.e., minimal sets of ABox assertions that (together with the TBox) yield the given answer. This can be useful for further computation of more complex explanations, e.g. proofs [22, 19, 23], since it can reduce large ABoxes to a handful of facts.

The runtime of this approach is substantially larger than what is required for query answering in the first place, which is mainly due to ProvSQL computing the provenance for all query answers at once. In practice, we expect that such explanations are usually only computed upon request by a user, and only for single answers at a time. We compare the performance of our approach with Soufflé, a Datalog engine that supports proofs for query atoms [19], from which one can extract a (possibly non-minimal) set of ABox assertions responsible for the query answer. In contrast to our approach, this only computes a single justification at a time, but still requires significant time. The data and instructions for running the experiments can be found at <https://lat.inf.tu-dresden.de/~stefborg/abox-justifications.zip>.

2. Preliminaries

We assume that the reader is familiar with the basics of DLs [25]. A *concept atom* is of the form $A(t)$, where A is a concept name and the *term* t is an individual name or a variable. Similarly, a *role atom* is of the form $r(t, t')$, where r is a role name and t, t' are terms. A Datalog atom $b(t_1, t_2, \dots, t_n)$ is a relational symbol b along with a list of terms as arguments. A Datalog program is a collection of Datalog rules, each of which is of the form:

$$h :- b_1, b_2, \dots, b_n$$

where $n \geq 0$, h, b_1, \dots, b_n are atoms, h is the *head* of the rule, and the conjunction of b_1, b_2, \dots, b_n is the *body* of the rule. The predicate in the rule head *depends on* each predicate occurring in the body. A program is recursive if the transitive closure of the “*depends on*”-relation is cyclic.

A *conjunctive query* (CQ) is a first-order formula of the form $q(\vec{x}) = \exists \vec{y}. \phi(\vec{x}, \vec{y})$, where ϕ is a conjunction of concept and role atoms using the variables in \vec{x} and \vec{y} . The variables in \vec{x} are the *answer variables* of q . An *answer* to q over an ontology $(\mathcal{A}, \mathcal{T})$ with ABox \mathcal{A} and TBox \mathcal{T} is a $|\vec{x}|$ -tuple \vec{a} of individual names from \mathcal{A} such that every model of $(\mathcal{A}, \mathcal{T})$ satisfies $q(\vec{a})$ under the usual semantics of first-order logic (written $\mathcal{A}, \mathcal{T} \models q(\vec{a})$). A *Datalog rewriting* of q w.r.t. \mathcal{T} is a tuple $(P_{q,\mathcal{T}}, Q)$, where $P_{q,\mathcal{T}}$ is a Datalog program with the $|\vec{x}|$ -ary *query predicate* Q , such that, for all ABoxes \mathcal{A} over the signature of q and \mathcal{T} , it holds that $\mathcal{A}, \mathcal{T} \models q(\vec{a})$ iff $\mathcal{A} \cup P_{q,\mathcal{T}} \models Q(\vec{a})$. Clipper is a tool that can compute Datalog rewritings for arbitrary CQs and TBoxes formulated in Horn-*SHIQ* [24].

Example 1. *In a scenario where a human operator oversees a collection of drones, each drone can be connected to an individual describing its surroundings via the role environment. Since sensor readings are often imperfect, the description can contain additional confidence information. Consider a CQ $q(x) = \exists y. \text{environment}(x, y) \wedge \text{LowVisiblity}(y) \wedge \text{HighConfidence}(y)$ over a TBox*

$$\begin{aligned} \text{WetDrone} &\sqsubseteq \exists \text{environment}. (\text{Rain} \sqcap \text{HighConfidence}) \\ \text{Rain} &\sqsubseteq \text{LowVisiblity} \end{aligned}$$

The query returns a set of individuals (drones) that are likely in a low-visibility environment. The ontology additionally specifies that rain decreases visibility and, if a drone is wet, it is likely raining. A Datalog rewriting of the query and ontology could look as follows.

$$\begin{aligned} \text{LowVisiblity}(x) &: \neg \text{Rain}(x) \\ Q(x) &: \neg \text{environment}(x, y), \text{LowVisiblity}(y), \text{HighConfidence}(y) \\ Q(x) &: \neg \text{WetDrone}(x) \end{aligned}$$

An *ABox justification* for a query answer $\mathcal{A}, \mathcal{T} \models q(\vec{a})$ is a minimal subset $\mathcal{J} \subseteq \mathcal{A}$ for which $\mathcal{J}, \mathcal{T} \models q(\vec{a})$ holds. Using a Datalog rewriting $(P_{q,\mathcal{T}}, Q)$, this can be reformulated to $\mathcal{J} \cup P_{q,\mathcal{T}} \models Q(\vec{a})$. The latter problem can be solved in the closed-world setting of databases, and therefore we will apply database provenance computation to solve it. The *why-provenance* for a database query answer $\mathcal{A} \models q(\vec{a})$ is the set of all minimal subsets of \mathcal{A} that yield the same answer. ProVSQL is a plugin for the Postgres relational database system [14] that can compute different kinds of provenance database queries, supporting most features of the SQL standard. Any CQ q over a database can be seen as an SQL query of the form `SELECT (DISTINCT) . . . FROM . . . WHERE . . .`.

In the setting of ontology-based data access, databases often come equipped with so-called *mappings* that associate data from the data sources with concepts in the ontology. In particular, they can define unary and binary relations in terms of SQL queries over an input database that may contain tables with more than two attributes. This type of mappings is usually referred to as *global-as-view* [26]. Formally, a mapping m has the form $m: \mathbf{q}(\vec{x}) \rightsquigarrow N(\vec{x})$, where \mathbf{q} is an SQL query over the data sources, and N is a DL concept or role name. Intuitively, \mathbf{q} returns (pairs of) constants from the data sources and instantiates N with them.

In this paper, we assume these mappings to be pre-materialized into unary and binary tables that correspond to DL concepts and roles, respectively. In some cases, splitting a large table into smaller ones also allows for more fine-grained justifications.

Example 2. Using mappings, a database entry of a (ProvSQL-enabled) *SensorData* table

<i>id</i>	<i>drone</i>	<i>sensor</i>	<i>environment</i>	<i>time</i>	<i>confidence</i>	<i>provsql</i>
123	<i>d2</i>	<i>camera</i>	<i>rain</i>	15:43	0.8	<i>a0eebc99-9c0b-4ef8</i>

can be transformed into separate facts *sensor*(123, *camera*), *Rain*(123), *time*(123, 15:43), *HighConfidence*(123), *environment*(*d2*, 123). Consider the query

$$q'(x) = \exists y. \text{environment}(x, y) \wedge \text{Rain}(y) \wedge \text{HighConfidence}(y)$$

that has the answer $x = d2$ w.r.t. the TBox from Example 1. The only justification for this answer is $\{\text{environment}(d2, 123), \text{Rain}(123), \text{HighConfidence}(123)\}$, which allows pinpointing the reasons for the query answer more precisely than by just returning the full original tuple *SensorData*(123, *d2*, *camera*, *rain*, 15:43, 0.8).

3. Implementation

In this section, we describe the building blocks of our approach for computing ABox justifications for CQ answers.

3.1. ProvSQL

ProvSQL² [14] is an open-source module for the PostgreSQL database management system that adds support for computation of provenance of (SQL) query results. The module computes the *provenance circuit* associated with a query that represents the operations performed to obtain a query answer, such as join \otimes , union \oplus , etc. It adds a separate column, *provsql*, that contains *provenance tokens*, to all ProvSQL-enabled tables of the database. Provenance tokens are fresh universally unique identifiers (UUIDs) and correspond to input gates of the circuits. ProvSQL generates new provenance tokens for results of queries on ProvSQL-aware tables, which identify inner gates of the provenance circuits.

Example 3. Returning back to Example 2 above, using ProvSQL, we can trace the DL-ified tuples back to the original *SensorData* table, since the facts such as *environment*(*d2*, 123) can be obtained by an SQL query over the materialized *environment* table:

<i>drone</i>	<i>id</i>	<i>provsql</i>
<i>d2</i>	123	<i>6bb9bd38-4ef8-bb6d</i>

where the UUID '*6bb9bd38-4ef8-bb6d*' refers to a circuit that encodes the projection operation over the original entry '*a0eebc99-9c0b-4ef8*'.

To obtain the set of all justifications $\mathfrak{J} = \{\mathcal{J}_1, \dots, \mathcal{J}_n\}$ for a specific output tuple, we then need to suitably evaluate its provenance circuit. Each UUID corresponding to an input fact f gets replaced by $\{\{f\}\}$ (the set containing only the trivial justification for f), and the operators \otimes , \oplus are translated into element-wise set union and simple set union, respectively, which corresponds to a step-wise combination of the sets of justifications for intermediate query results [13, 16]. Afterwards, we still need to remove non-minimal elements from the set \mathfrak{J} , but this was mostly unnecessary in our experiments.

²<https://github.com/PierreSenellart/provsql>

3.2. Clipper

Clipper³ is DL reasoner for conjunctive query answering over Horn- \mathcal{SHIQ} ontologies via Datalog rewriting that is implemented in Java. It accepts an ontology $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ and a CQ q in the SPARQL syntax as input and produces a Datalog program which can be evaluated over the facts in \mathcal{A} . In the original paper [24], the authors use DLV⁴ [27] or Clingo⁵ [28] for the final Datalog evaluation. The program comprises rewriting steps of the query q as well as the completion rules over the TBox \mathcal{T} . In general, the resulting Datalog program can be recursive.

Example 4. *The axiom $\exists \text{near}.\exists \text{environment}.\text{Rain} \sqsubseteq \exists \text{environment}.\text{Rain}$, expressing that, if rain was detected at a nearby location, then it is also raining at the current location, will be rewritten as*

$$\begin{aligned} \text{environmentSomeRain}(x) &: -\text{near}(x, y), \text{environmentSomeRain}(y) \\ \text{environmentSomeRain}(x) &: -\text{environment}(x, y), \text{Rain}(y) \end{aligned}$$

In the next section, we explain how we can transform a possibly recursive Datalog program into a series of SQL statements, whose provenance can be traced using ProVSQL.

3.3. Compiling Datalog Rules Into SQL Statements

The algorithm proceeds as follows. First, the “*depends on*”-relation of the Datalog program is represented as a graph, denoted by $G = (V, E)$, where each node n_p in V corresponds to a predicate p that occurs in the ontology. The edge relation E is defined using the body and the head of each rule: for each rule $h :- b_1, \dots, b_n$, there exist edges $(n_h, n_{b_1}), \dots, (n_h, n_{b_n})$ in E . Furthermore, each node n_p is associated with the set of all rules in the Datalog program that have p in their head. In Datalog rewritings produced by Clipper, no other predicate depends on the query predicate Q , and thus for most of the following descriptions we will ignore the node n_Q and process it separately at the end.

If G is acyclic, its nodes can be partitioned into *levels* according to the length of the longest path to any leaf node (a node without outgoing edges). The algorithm processes all nodes on the same level one after the other, starting from the lowest level, i.e., the leafs. Processing a node n translates all rules associated with n (which all have the same head predicate) into SQL statements.

Example 5. *For example, the rules $h(x_1) :- b_1(x_1, x_2), b_2(x_2)$, $h(x_1) :- b_3(x_1)$ would be translated into the query*

```
CREATE TABLE h AS
((SELECT b1.x1 FROM b1 JOIN b2 ON b1.x2 = b2.x1) UNION (SELECT x1 FROM b3)).
```

In case a table for a predicate h already exists and has to be updated with new tuples, in order for ProVSQL to recognize a change in the data, first a dummy table needs to be created that holds the union of the data contained in the old table and the new query results:

```
CREATE TABLE dummy AS (. . . UNION SELECT * FROM h)
```

³<https://github.com/ghxiao/clipper>

⁴<https://www.dlvsystem.it/dlvsite/>

⁵<https://github.com/potassco/clingo>

Afterwards, the old table is deleted and re-created with the content of the dummy table:

```
DROP TABLE h
CREATE TABLE h AS SELECT * FROM dummy
```

However, the dependency graph of the rules created by Clipper from a Horn-*SHIQ* ontology can actually be cyclic (see Example 4). To handle cyclic dependency graphs, all strongly connected components (SCCs) are first identified using Kosaraju-Sharir’s algorithm [30], which is implemented in the `jgraph`⁶ library. All nodes in a given SCC are then replaced by a new *cycle node* in the dependency graph.

For an SCC S and a new cycle node n , the dependency graph $G = (V, E)$ is updated to $G' = (V', E')$ with $V' = (V \setminus S) \cup \{n\}$ and

$$\begin{aligned}
 E' = & \{(a, b) \mid (a, b) \in E, a, b \in V'\} \cup \\
 & \{(a, n) \mid (a, b) \in E, a \in V', b \in S\} \cup \\
 & \{(n, b) \mid (a, b) \in E, a \in S, b \in V'\}.
 \end{aligned} \tag{1}$$

The rules previously associated to the nodes in S are thereafter associated to n . The algorithm then proceeds as in the non-recursive case, but processes cycle nodes differently, depending on the associated rules. Due to the shape of the Datalog programs produced by Clipper, all rules involved in a cycle either have only unary predicates in the head (e.g. as in Example 4), or they have only binary predicates in the head [24]. In the former case, to process the cycle node, first the length ℓ of the longest acyclic path of the SCC is determined. Then, all SQL statements of the rules associated with the cycle node are executed ℓ times to ensure that all individuals are propagated to all involved predicates.

By Horn-*SHIQ* syntax, when binary head atoms are involved, all rules must be of one of the following shapes:

$$r(x, y) :- s(x, y) \tag{2}$$

$$r(x, y) :- s(y, x) \tag{3}$$

$$r(x, z) :- r(x, y), r(y, z) \tag{4}$$

If all rules in the cycle are of the form (2), they can be processed as in the case of unary head atoms. If rules of shape (3) are involved, we need to double the number of executions of the SQL statements since it may require 2ℓ iterations to propagate all involved pairs of individuals (a, b) and their inverses (b, a) to all involved binary predicates.

Finally, transitivity rules of the form (4) need to be processed differently, since their application involves a potentially unbounded number of individuals, and thus the necessary number of iterated SQL statements now depends on the length of the longest (undirected) path in the current database that involves only the roles from the cycle node. Therefore, every transitivity rule (potentially as part of a larger cycle node) increases the number of iterated applications of SQL statements for the whole cycle node by $\log_2 k$, where k is the length of this path.

At the very end of the algorithm, the query predicate Q is populated using the same approach as in Example 5, since we do not have to worry about cycles at this stage.

Algorithm 1: Generating SQL updates w.r.t. a Datalog program P for ABox tables \mathcal{A} .

```

1  $G = (V, E)$  is the “depends-on” relation graph of  $P$ 
2 foreach  $n_p \in V$  do
3   |  $\text{rule}(n_p) = \{\rho \in P \mid p \text{ is in the head of } \rho\}$ 
4  $G' = G$ 
5 while  $G'$  has a SCC  $S$  do
6   | create a new cycle node  $n_S$ 
7   |  $\text{rule}(n_S) = \{\rho \in P \mid \rho \in \text{rule}(v) \text{ for some } v \in S\}$ 
8   | update  $G'$  according to (1) for  $n_S$ 
9  $\mathcal{A}' = \mathcal{A}$ 
10 repeat
11   |  $N = \text{leaf}(G') = \{v \in V' \mid \text{there is no } n \in V' \text{ such that } (v, n) \in E'\}$ 
12   | foreach  $v \in N$  do
13     | if  $v = n_S$  for some SCC  $S$  in  $G'$  then
14       |  $\ell = \text{length of the longest acyclic path in } S$ 
15       | if there is  $\rho \in \text{rule}(n_S)$  of the form (3) then
16         |  $\ell = 2\ell$ 
17       | if there is  $\rho \in \text{rule}(n_S)$  of the form (4) then
18         |  $\mathcal{P} = \{p \mid p \text{ occurs in the body of some } \rho \in \text{rule}(n_S)\}$ 
19         |  $k = \text{length of the longest path in } \mathcal{A}' \text{ involving only predicates from } \mathcal{P}$ 
20         |  $\ell = \ell + \log_2 k$ 
21       | else
22         |  $\ell = 1$ 
23       |  $U = \text{set of SQL statements for } \text{rule}(v)$  (e.g. see Example 5)
24       | apply  $U$  to  $\mathcal{A}'$   $\ell$  times
25       | remove  $v$  from  $G'$ 
26 until  $G'$  is empty
27 return  $\mathcal{A}'$ 

```

The above-described procedure is summarized in Algorithm 1.

Theorem 1. Let $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ be a Horn-SHIQ ontology, q a CQ with Datalog rewriting $(P_{q,\mathcal{T}}, Q)$ w.r.t. \mathcal{T} , and \mathcal{A}' the set of facts resulting from Algorithm 1 w.r.t. the Datalog program $P_{q,\mathcal{T}}$ and \mathcal{A} . Then $\mathcal{O} \models q(\vec{a})$ iff $Q(\vec{a}) \in \mathcal{A}'$.

Proof. For soundness, it suffices to observe that the algorithm faithfully applies the rules in the rewriting, which is correct for query answering over \mathcal{O} . To show completeness, we have to show that $\mathcal{A} \cup P_{q,\mathcal{T}} \models Q(\vec{a})$ implies $Q(\vec{a}) \in \mathcal{A}'$. If $\mathcal{A} \cup P_{q,\mathcal{T}} \models Q(\vec{a})$, there exists a derivation \mathfrak{D} (i.e., a tree of rule applications) deriving $Q(\vec{a})$ from the facts in \mathcal{A} using the rules in $P_{q,\mathcal{T}}$. In the following, we assume that \mathfrak{D} is minimal, and in particular that no fact is derived twice and only facts that are required to derive $Q(\vec{a})$ are derived. For every fact $P(\vec{c})$ derived in \mathfrak{D} , we prove

⁶<https://github.com/jgraph>

that $P(\vec{c}) \in \mathcal{A}'$ by induction over the order in which the node n representing P is processed by the main loop in Lines 11–25. Hence, we assume that the claim is satisfied for all predicates represented by the nodes reachable from n in the original graph G' (after the introduction of the cycle nodes in Lines 5–8). We distinguish two cases.

- If n is an ordinary node n_P or a cycle node associated with only unary or only binary head predicates (including P), but rules of the form (3) or (4), then we consider the last rule applications in \mathfrak{D} involving the predicates in n . For any premises $P'(\vec{c}')$ of these rules involving predicates from previous stages, we know by induction that $P'(\vec{c}') \in \mathcal{A}'$ has already been derived previously by Line 24. The remaining premises must be of the form $P'(\vec{c})$ with P' represented by the current cycle node n . Assume that the depth of these last rule applications is larger than the length ℓ of the longest acyclic path in the SCC represented by n . Then the rule applications contain a cycle, i.e., at least one fact $P'(\vec{c})$ is derived twice, which contradicts our assumption on the derivation \mathfrak{D} . Thus, since the depth of the last rule applications to derive $P(\vec{c})$ is bounded by ℓ , we have $P(\vec{c}) \in \mathcal{A}'$ by our construction.
- If n involves binary predicates and at least one rule of the form (3), but no rule of the form (4), then we can apply the same arguments as above, but have to consider all facts of the form $P'(c, d)$ and $P'(d, c)$ that are involved in the derivation of $P(\vec{c}) = P(c, d)$, which means that the length of the derivation could be doubled in the worst case (see Line 16).
- If n is a cycle node involving a transitive binary predicate r (cf. Lines 17–20), then $P(\vec{c}) = P(c, d)$ and we consider all applications of the rule $r(x, z) \leftarrow r(x, y), r(y, z)$ up to the derivation of $r(c, d)$ (or $r(d, c)$). Since \mathfrak{D} is non-redundant, the original r -facts (not derived by the transitivity rule) must form an r -path $r(c, c_1), r(c_1, c_2), \dots, r(c_n, d)$ from c to d without repeating constants. If this path is of length k , then the fact $r(c, d)$ is derived after at most $\log_2 k$ nested applications of $r(x, z) \leftarrow r(x, y), r(y, z)$. By similar arguments as above, each of the r -facts $r(c_i, c_{i+1})$ must have been derived after 2ℓ rule applications from other binary facts (or their inverses) whose predicates are involved in the cycle. This shows that $r(c, d) \in \mathcal{A}'$ and $P(c, d) \in \mathcal{A}'$. \square

Note that Algorithm 1 does consider the query predicate Q : the ABox tables are updated until no more facts are derived. In the implementation, this procedure is optimized by terminating once the node n_Q has been processed by the loop in Lines 12–25. Further iterations are redundant, since they cannot create new facts for Q .

4. Evaluation

In order to evaluate our approach, we compare our implementation with Soufflé, a state-of-the-art tool for Datalog reasoning that can also compute proofs [19], on the Lehigh University Benchmark (LUBM).⁷ All experiments were run on a computer with an Intel Core i5-7200U CPU @ 2.50GHz processor, and runtime and memory cutoffs of 14400 seconds and 2 GB. The experiment results are shown in Table 1.

⁷<http://swat.cse.lehigh.edu/projects/lubm/>

Table 1

Comparing Soufflé (s) and our approach (o) with and without (-) provenance over the LUBM queries.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14
#ans	4	0	6	34	719	7790	67	7790	208	4	224	15	1	5916
<u>Query Time (s)</u>														
s	1.6	2.2	3.0	-	-	3.1	5.4	2.8	4.5	2.3	0	2.2	0	2.2
s-	0.7	0.5	0.3	1.8	1.3	1.1	2.3	1.7	2.1	1.1	0	1.3	0	1.3
o	25	19	83	102	326	343	935	87	1251	4968	18	43	80	80
o-	17	15	17	14	16	13	16	16	16	17	17	35	12	17
<u>Explanation Time (s)</u>														
s	0.03	-	244	-	-	0.1	326	51	0.002	1381	0.2	0.08	0.01	0.01
o	0.1	-	0.05	0.2	237	318	908	65	1226	4939	0.05	0.05	0.06	0.08
<u>Total Time (s)</u>														
s	1.6	-	247	-	-	3.2	331	54	4.5	1383	0.2	2.2	0	2.2
o	25	-	83	102	563	661	1843	152	2477	9907	18	43	80	80

We also wanted to compare the performance of a recent approach for computing provenance for Datalog programs based on Rulewerk and Vlog [31], but we could not achieve reasonable runtimes with it. We also did not compare with OntoProv [20], which also uses ProVSQL, but only supports *DL-Lite* ontologies. Using Clipper, the benchmark’s University ontology was translated into a Datalog program, for which we ran the two tools together with a dataset containing the data of one university. We recorded how long it took to answer each of the 14 benchmark queries (Query Time), as well as the time needed for computing justifications for a single (randomly chosen) query answer (Explanation Time). Query 2 has no answers, which is why we cannot compute any justifications for it. For Queries 4 and 5, Soufflé did not produce a result in the limited time. Additionally, we provide the Query Time without the provenance computation in the lines marked by s- and o-.

The Query Time for our approach already includes the time taken by ProVSQL to compute the provenance circuits for each intermediate answer, which could also be seen as part of the Explanation Time, which is why we also report the Total Time for each tool. Nevertheless, the comparison is still biased since ProVSQL always computes the full provenance, i.e., we obtain *all possible* justifications for a query answer, whereas Soufflé outputs only a single proof, from which we can extract one (possibly non-minimal) set of input facts responsible for the answer. By construction, our approach cannot compute the provenance of just a single query answer on demand without pre-computing the provenance for all answers. Similarly, ProVSQL does not support computing only a single ABox justification.

We can see that with the provenance components our algorithm mostly compares favorably to Soufflé, except for queries that have a lot of answers (which requires ProVSQL to compute many provenance circuits) and for answers whose provenance is very large due to the cycles in the Datalog program. Since we only want to compute justifications, these provenance circuits contain a lot of redundancies that could be eliminated at intermediate steps by a more dedicated algorithm in the future.

5. Conclusion

We presented a possible approach for computing ABox justifications, or why-provenance, for relatively expressive ontology-mediated queries that have Datalog rewritings. This can be useful to compute further, more detailed explanations of query answers by restricting the set of facts needed for computing them [22, 19, 32]. Similarly, to compute more detailed proofs, it would be interesting to directly explain the rewriting steps used by Clipper [22, 23].

Acknowledgments

We thank Islam Hamada for his contributions to the implementation. This work was supported by the DFG grant 389792660 as part of TRR 248 (<https://perspicuous-computing.science>).

References

- [1] S. Schlobach, R. Cornet, Non-standard reasoning services for the debugging of description logic terminologies, in: G. Gottlob, T. Walsh (Eds.), IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, 2003, pp. 355–362. URL: <http://ijcai.org/Proceedings/03/Papers/053.pdf>.
- [2] F. Baader, R. Peñaloza, B. Suntisrivaraporn, Pinpointing in the description logic EL^+ , in: J. Hertzberg, M. Beetz, R. Englert (Eds.), KI 2007: Advances in Artificial Intelligence, 30th Annual German Conference on AI, volume 4667 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 52–67. doi:10.1007/978-3-540-74565-5_7.
- [3] M. Horridge, Justification Based Explanation in Ontologies, Ph.D. thesis, University of Manchester, UK, 2011. URL: https://www.research.manchester.ac.uk/portal/files/54511395/FULL_TEXT.PDF.
- [4] N. Manthey, R. Peñaloza, S. Rudolph, SATPin: Axiom pinpointing for lightweight description logics through incremental SAT, *Künstliche Intell.* 34 (2020) 389–394. doi:10.1007/s13218-020-00669-4.
- [5] M. Horridge, B. Parsia, U. Sattler, Explaining inconsistencies in OWL ontologies, in: L. Godo, A. Pugliese (Eds.), Scalable Uncertainty Management, Third International Conference, SUM, Proceedings, volume 5785 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 124–137. doi:10.1007/978-3-642-04388-8_11.
- [6] A. Borgida, E. Franconi, I. Horrocks, Explaining ALC subsumption, in: W. Horn (Ed.), ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, IOS Press, 2000, pp. 209–213. URL: <http://www.frontiersinai.com/ecai/ecai2000/pdf/p0209.pdf>.
- [7] D. L. McGuinness, Explaining Reasoning in Description Logics, Ph.D. thesis, Rutgers University, NJ, USA, 1996. doi:10.7282/t3-q0c6-5305.
- [8] M. Horridge, B. Parsia, U. Sattler, Justification oriented proofs in OWL, in: P. F. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Z. Pan, I. Horrocks, B. Glimm (Eds.), The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, Revised Selected Papers, Part I, volume 6496 of *Lecture Notes in Computer Science*, Springer, 2010, pp. 354–369. doi:10.1007/978-3-642-17746-0_23.

- [9] C. Alrabbaa, F. Baader, S. Borgwardt, P. Koopmann, A. Kovtunova, Finding small proofs for description logic entailments: Theory and practice, in: E. Albert, L. Kovács (Eds.), LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, volume 73 of *EPiC Series in Computing*, EasyChair, 2020, pp. 32–67. doi:10.29007/nhpp.
- [10] C. Alrabbaa, F. Baader, S. Borgwardt, P. Koopmann, A. Kovtunova, Finding good proofs for description logic entailments using recursive quality measures, in: A. Platzer, G. Sutcliffe (Eds.), Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, volume 12699 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 291–308. doi:10.1007/978-3-030-79876-5_17.
- [11] C. Alrabbaa, F. Baader, S. Borgwardt, R. Dachsel, P. Koopmann, J. Méndez, Evonne: Interactive proof visualization for description logics (system description), in: J. Blanchette, L. Kovács, D. Pattinson (Eds.), Automated Reasoning - 11th International Joint Conference, IJCAR 2022, Proceedings, volume 13385 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 271–280. doi:10.1007/978-3-031-10769-6_16.
- [12] J. Cheney, L. Chiticariu, W. C. Tan, Provenance in databases: Why, how, and where, *Found. Trends Databases* 1 (2009) 379–474. doi:10.1561/19000000006.
- [13] P. Senellart, Provenance in databases: Principles and applications, in: M. Krötzsch, D. Stepanova (Eds.), Reasoning Web. Explainable Artificial Intelligence - 15th International Summer School 2019, Tutorial Lectures, volume 11810 of *Lecture Notes in Computer Science*, Springer, 2019, pp. 104–109. doi:10.1007/978-3-030-31423-1_3.
- [14] P. Senellart, L. Jachiet, S. Maniu, Y. Ramusat, ProvSQL: Provenance and probability management in PostgreSQL, *Proc. VLDB Endow.* 11 (2018) 2034–2037. doi:10.14778/3229863.3236253.
- [15] T. J. Green, G. Karvounarakis, V. Tannen, Provenance semirings, in: L. Libkin (Ed.), Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China, ACM, 2007, pp. 31–40. doi:10.1145/1265530.1265535.
- [16] P. Buneman, S. Khanna, W. C. Tan, Why and where: A characterization of data provenance, in: J. V. den Bussche, V. Vianu (Eds.), Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings, volume 1973 of *Lecture Notes in Computer Science*, Springer, 2001, pp. 316–330. doi:10.1007/3-540-44503-X_20.
- [17] Y. Ramusat, S. Maniu, P. Senellart, Efficient provenance-aware querying of graph databases with Datalog, in: V. Kalavri, S. Salihoglu (Eds.), GRADES-NDA '22: Proceedings of the 5th ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA), ACM, 2022, pp. 4:1–4:9. doi:10.1145/3534540.3534689.
- [18] C. Bourgaux, P. Bourhis, L. Peterfreund, M. Thomazo, Revisiting semiring provenance for datalog, in: G. Kern-Isberner, G. Lakemeyer, T. Meyer (Eds.), Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR, 2022. URL: <https://proceedings.kr.org/2022/10/>.
- [19] D. Zhao, P. Subotic, B. Scholz, Debugging large-scale Datalog: A scalable provenance evaluation strategy, *ACM Trans. Program. Lang. Syst.* 42 (2020) 7:1–7:35. doi:10.1145/3379446.

- [20] D. Calvanese, D. Lanti, A. Ozaki, R. Peñaloza, G. Xiao, Enriching ontology-based data access with provenance, in: S. Kraus (Ed.), Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI, ijcai.org, 2019, pp. 1616–1623. doi:10.24963/ijcai.2019/224.
- [21] C. Bourgaux, A. Ozaki, R. Peñaloza, L. Predoiu, Provenance for the description logic ELHr, in: C. Bessiere (Ed.), Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, ijcai.org, 2020, pp. 1862–1869. doi:10.24963/ijcai.2020/258.
- [22] A. Borgida, D. Calvanese, M. Rodriguez-Muro, Explanation in the DL-Lite family of description logics, in: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems: OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Proceedings, Part II, volume 5332 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 1440–1457. doi:10.1007/978-3-540-88873-4_35.
- [23] C. Alrabbaa, S. Borgwardt, P. Koopmann, A. Kovtunova, Explaining ontology-mediated query answers using proofs over universal models, in: G. Governatori, A. Turhan (Eds.), Rules and Reasoning - 6th International Joint Conference on Rules and Reasoning, RuleML+RR 2022, Proceedings, volume 13752 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 167–182. doi:10.1007/978-3-031-21541-4_11.
- [24] T. Eiter, M. Ortiz, M. Simkus, T. Tran, G. Xiao, Query rewriting for Horn-SHIQ plus rules, in: J. Hoffmann, B. Selman (Eds.), Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22–26, 2012, Toronto, Ontario, Canada, AAAI Press, 2012. doi:10.1609/aaai.v26i1.8219.
- [25] F. Baader, I. Horrocks, C. Lutz, U. Sattler, An Introduction to Description Logic, Cambridge University Press, 2017. doi:10.1017/9781139025355.
- [26] M. Lenzerini, Data integration: A theoretical perspective, in: L. Popa, S. Abiteboul, P. G. Kolaitis (Eds.), Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3–5, Madison, Wisconsin, USA, ACM, 2002, pp. 233–246. doi:10.1145/543613.543644.
- [27] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, F. Scarcello, The DLV system for knowledge representation and reasoning, *ACM Trans. Comput. Log.* 7 (2006) 499–562. doi:10.1145/1149114.1149117.
- [28] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, M. Schneider, Potassco: The potsdam answer set solving collection, *AI Commun.* 24 (2011) 107–124. doi:10.3233/AIC-2011-0491.
- [29] C. Lutz, D. Toman, F. Wolter, Conjunctive query answering in the description logic EL using a relational database system, in: C. Boutilier (Ed.), IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11–17, 2009, 2009, pp. 2070–2075. URL: <http://ijcai.org/Proceedings/09/Papers/341.pdf>.
- [30] M. Sharir, A strong-connectivity algorithm and its applications in data flow analysis, *Computers & Mathematics with Applications* 7 (1981) 67–72. doi:[https://doi.org/10.1016/0898-1221\(81\)90008-0](https://doi.org/10.1016/0898-1221(81)90008-0).
- [31] A. Elhalawati, M. Krötzsch, S. Mennicke, An existential rule framework for computing why-provenance on-demand for Datalog, in: G. Governatori, A. Turhan (Eds.), Rules and Reasoning - 6th International Joint Conference on Rules and Reasoning, RuleML+RR,

Proceedings, volume 13752 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 146–163. doi:10.1007/978-3-031-21541-4_10.

- [32] C. Alrabbaa, S. Borgwardt, A. Hirsch, N. Knieriemen, A. Kovtunova, A. M. Rothermel, F. Wiehr, In the head of the beholder: Comparing different proof representations, in: G. Governatori, A. Turhan (Eds.), *Rules and Reasoning - 6th International Joint Conference on Rules and Reasoning, RuleML+RR 2022*, Berlin, Germany, September 26-28, 2022, Proceedings, volume 13752 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 211–226. doi:10.1007/978-3-031-21541-4_14.