Pay-as-you-go Ontology Query Answering Using a Datalog Reasoner*

Yujiao Zhou, Yavor Nenov, Bernardo Cuenca Grau, and Ian Horrocks

Department of Computer Science, University of Oxford, UK

Abstract. We describe a hybrid approach to conjunctive query answering over OWL 2 ontologies that combines a datalog reasoner with a fully-fledged OWL 2 reasoner in order to provide scalable "pay as you go" performance. Our approach delegates the bulk of the computation to the highly scalable datalog engine and resorts to expensive OWL 2 reasoning only as necessary to fully answer the query. We have implemented a prototype system that uses RDFox as a datalog reasoner, and HermiT as an OWL 2 reasoner. Our evaluation over both benchmark and realistic ontologies and datasets suggests the feasibility of our approach.

1 Introduction

The use of RDF [15], OWL 2 [16], and SPARQL 1.1 [25] to represent and query semi-structured data together with domain knowledge is increasingly widespread. Query answering in this setting is, however, of high worst-case complexity [7, 5], and although heavily optimised, existing systems for query answering w.r.t. RDF data and an unrestricted OWL 2 ontology can process only small to medium size datasets [13, 26, 11]. This has led to the development of query answering procedures that are more scalable, but that can (fully) process only fragments of OWL 2, and several prominent fragments have now been standardised as OWL 2 profiles [14]. Such systems have been shown to be (potentially) highly scalable [24, 1, 23, 27], but if the ontology falls outside the relevant profile, then the answers computed by such a system may be incomplete: if it returns an answer, then all tuples in the answer are (usually) valid, but some valid tuples may be missing from the answer. When used with out-of-profile ontologies, a query answer computed by such a system can thus be understood as providing a *lower-bound* on the correct answer; however, they cannot in general provide any upper bound or even any indication as to how complete the computed answer is [2].

In this paper, we describe a hybrid approach to query answering that exploits a datalog reasoner to compute both a lower bound answer and an upper bound answer. If lower and upper bound answers coincide, they obviously provide a sound and complete answer. Otherwise, *relevant fragments* of the ontology and data can be extracted that are guaranteed to be sufficient to test the validity of tuples in the "gap" between the two answers. These fragments can also be

* This paper recapitulates some of our results in [28–30], and it is accompanied with an appendix containing all proofs.

computed by relying solely on the datalog reasoner, and are typically much smaller than the input ontology and data. The remaining gap tuples need to be checked w.r.t. to the identified fragments using an OWL 2 reasoner such as HermiT [17] or Pellet [19]; furthermore, since the number of gap tuples can be significant in some cases, we exploit summarisation techniques inspired by the SHER system [3, 4] to quickly identify spurious gap answers, thus further reducing the requirement for fully-fledged OWL 2 reasoning.

Our approach is *pay-as-you-go* in the sense that the bulk of the computation is delegated to a scalable datalog engine. Furthermore, although our main goal is to answer queries over OWL 2 ontologies efficiently, our technical results are very general and our approach is not restricted to DLs. More precisely, given a first-order KR language L that can be captured by rules allowing for existential quantification and disjunction in the head, and over which we want to answer conjunctive queries, our only assumption is the availability of a fully-fledged reasoner for L and a datalog reasoner, which are both used as a "black box".

We have implemented our techniques in a prototypical system using the RDFox as a datalog reasoner [24] and the HermiT as a fully-fledged OWL 2 reasoner.[1] Our preliminary evaluation over both benchmark and realistic data suggests that the system can provide scalable pay-as-you-go query answering for a wide range of OWL 2 ontologies, RDF data and queries. In almost all cases, the system is able to completely answer queries without resorting to fully-fledged OWL 2 reasoning, and even when this is not the case, relevant fragment extraction and summarisation are effective in reducing the size of the problem to manageable proportions.

2 Preliminaries

We adopt standard first order logic notions, such as variables, constants, atoms, formulas, clauses, substitutions, satisfiability, and entailment. We also assume basic familarity with OWL 2 [16] and its profiles [14]. A generalised rule (or just a *rule*) is a function-free sentence of the form

$$\forall \mathbf{x}\,(\bigwedge_{j=0}^{n} B_j(\mathbf{x}) \rightarrow \bigvee_{i=0}^{m} \exists \mathbf{y}_i\, \varphi_i(\mathbf{x}, \mathbf{y}_i))$$

where $B_j(\mathbf{x})$ are *body* atoms and φ_i are conjunctions of *head* atoms. The universal quantifiers are left implicit from now on. A rule is *Horn* if $m \leq 1$, and it is *datalog* if it is Horn and does not contain existential quantifiers. A *fact* is a ground atom and a *dataset* is a finite set of facts. A *knowledge base* $\mathcal{K}$ consists of a finite set of rules and a dataset. We treat equality ($\approx$) as an ordinary predicate, but assume that every knowledge base in which equality occurs contains the axioms of equality for its signature. Each OWL 2 ontology can be normalised as one

[1] Although our techniques are proved correct for general conjunctive queries, in practice we are limited by the current query capabilities of OWL 2 reasoners.

$$\mathsf{Foreman}(x) \rightarrow \mathsf{Manag}(x) \qquad (T_1)$$
$$\mathsf{Superv}(x) \rightarrow \mathsf{Manag}(x) \qquad (T_2)$$
$$\mathsf{Superv}(x) \wedge \mathsf{boss}(x,y) \rightarrow \mathsf{Workman}(y) \qquad (T_3)$$
$$\mathsf{TeamLead}(x) \wedge \mathsf{boss}(x,y) \wedge \mathsf{Manag}(y) \rightarrow \qquad (T_4)$$
$$\mathsf{Manag}(x) \rightarrow \mathsf{Superv}(x) \vee \exists y.(\mathsf{boss}(x,y) \wedge \mathsf{Manag}(y)) \qquad (T_5)$$
$$\mathsf{Manag}(x) \rightarrow \exists y.(\mathsf{boss}(x,y)) \qquad (T_6)$$

$\mathsf{Manag}(Sue)$	(D_1)	$\mathsf{Superv}(Rob)$	(D_3)	$\mathsf{Manag}(Jo)$	(D_5)
$\mathsf{Superv}(Dan)$	(D_2)	$\mathsf{boss}(Dan, Ben)$	(D_4)	$\mathsf{TeamLead}(Jo)$	(D_6)
				$\mathsf{boss}(Jane, Rob)$	(D_7)

Fig. 1. Example knowledge base $\mathcal{K}^{ex}$.

such knowledge base using the correspondence of OWL and first order logic and a variant of the structural transformation (e.g., see [17] for detais).

We focus on CQ answering as the key reasoning problem. A *query* is a formula $q(\mathbf{x}) = \exists \mathbf{y}\, \varphi(\mathbf{x}, \mathbf{y})$ with $\varphi(\mathbf{x}, \mathbf{y})$ a conjunction of atoms. We usually omit the free variables $\mathbf{x}$ of queries and write just q. The query is *atomic* if $\varphi(\mathbf{x}, \mathbf{y})$ is a single atom. A tuple of individuals $\mathbf{a}$ is a *(certain) answer* to q w.r.t. a set of sentences $\mathcal{F}$ iff $\mathcal{F} \models q(\mathbf{a})$. The set of all answers to $q(\mathbf{x})$ w.r.t. $\mathcal{F}$ is denoted by $\mathsf{cert}(q, \mathcal{F})$.

There are two main techniques for answering queries over a datalog knowledge base $\mathcal{K}$. *Forward chaining* computes the set $Mat(\mathcal{K})$ of ground atoms entailed by $\mathcal{K}$, called the *materialisation* of $\mathcal{K}$. A query q over $\mathcal{K}$ can be answered directly over the materialisation. *Backward chaining* treats a query as a conjunction of atoms (a *goal*). An *SLD resolvent* of a goal $A \wedge \psi$ with a datalog rule $\varphi \rightarrow C_1 \wedge \cdots \wedge C_n$ is a goal $\psi\theta \wedge \varphi\theta$, with θ the MGU of A and C_j, for some $1 \leq j \leq n$. An *SLD proof* of a goal G_0 in $\mathcal{K}$ is a sequence of goals $(G_0, \ldots, G_n)$ with G_n the empty goal ($\square$), and each G_{i+1} a resolvent of G_i and a rule in $\mathcal{K}$.

3 Overview

The main idea behind our approach to query answering is to delegate the bulk of the computational workload to a highly scalable datalog reasoner, thus minimising the use of a fully-fledged OWL 2 reasoner. Given a knowledge base $\mathcal{K}$ and a query q, we proceed according to the following algorithm:

1. Use the datalog reasoner to compute both lower bound (sound but possibly incomplete) and upper bound (complete but possibly unsound) answers to the (Boolean) unsatisfiability query and the input q. (See Sections 4, 5).
2. If both bounds report unsatisfiability, then we return unsatisfiable. If none of them reports unsatisfiability and they yield the same answers to q, we output the resulting answers. In any other case, proceed to the next step.

3. Use the datalog reasoner to compute fragments $\mathcal{K}_\bot$ and $\mathcal{K}_{[q,G]}$ of $\mathcal{K}$, where G is the set of answers to q in the gap between the bounds. (See Section 6).
4. If the upper bound reports unsatisfiability and $\mathcal{K}_\bot$ is unsatisfiable, then return unsatisfiable.
5. Use the OWL reasoner to check whether $\mathcal{K}_{[q,G]} \cup \mathcal{K}_\bot \models q(\mathbf{a})$, for each $\mathbf{a} \in G$. To minimise the computational workload of the OWL reasoner, this step is carried out as follows (see Section 7):
   (a) Summarise $\mathcal{K}_{[q,G]} \cup \mathcal{K}_\bot$ by merging all constants that instantiate the same unary predicates [4]. Use the OWL reasoner to discard those $\mathbf{a} \in G$ such that $q(\mathbf{a})$ is not entailed by the summarised KB.
   (b) Compute a dependency relation between the remaining elements of G such that if $\mathbf{b}$ depends on $\mathbf{a}$ and $\mathbf{a}$ is a spurious answer, then so is $\mathbf{b}$. Arrange the calls to the reasoners according to these dependencies.
6. Return the lower bound answers to q plus those tuples in G determined to be answers in Step 5.

We will describe each of these steps and illustrate them using as running example the knowledge base $\mathcal{K}^{ex}$ in Figure 1 and the following query q^{ex}:

$$q^{ex}(x) = \exists y(\mathsf{boss}(x,y) \land \mathsf{Workman}(y))$$

4 Computing Upper Bounds

To compute upper bound query answers, we first compute a datalog knowledge base $\mathcal{U}(\mathcal{K})$ that entails the nullary predicate $\bot$, if $\mathcal{K}$ is unsatisfiable, and that entails $\mathcal{K}$, otherwise. Hence, for satisfiable knowledge bases $\mathcal{K}$ we get that $\mathsf{cert}(q, \mathcal{U}(\mathcal{K}))$ subsumes $\mathsf{cert}(q, \mathcal{K})$. The knowledge base $\mathcal{U}(\mathcal{K})$ is the result of consecutively applying the transformations Σ, Ξ and Ψ defined next.

Definition 1. *Let $\mathcal{K}$ be a KB. We define $\mathcal{U}(\mathcal{K}) := \Psi \circ \Xi \circ \Sigma(\mathcal{K})$, where*

- *Σ is a mapping that transforms each rule into clausal normal form;*
- *Ξ maps each clause C to a set of clauses as follows:* (i) *if C contains only negative literals, then $\Xi(C) = C \lor \bot$;* (ii) *if C is of the form $\neg B_0 \lor \cdots \lor \neg B_k \lor C_0 \lor \cdots \lor C_{r+1}$ then $\Xi(C)$ consists of the clauses $\neg B_1 \lor \cdots \lor \neg B_k \lor C_i$, for $0 \le i \le r+1$;* (iii) *in any other case, $\Xi(C) = C$.*
- *Ψ maps every Horn clause C to a datalog rule $\Psi(C)$ obtained from C by first replacing each functional term with a globally fresh constant, and then transforming the resulting clause into its equivalent datalog rule.*

These transformations extend to sets in the natural way.

In our example $\mathcal{K}^{ex}$, the transformation $\mathcal{U}$ is the identity for all rules except T_4–T_6. Rule T_4 is transformed by $\mathcal{U}$ (Ξ in particular) into the datalog rule U_4.

$$\mathsf{TeamLead}(x) \land \mathsf{boss}(x,y) \land \mathsf{Manag}(y) \to \bot \qquad (U_4)$$

T_5 is first transformed by Σ into clauses $\neg\mathsf{Manag}(x) \vee \mathsf{Superv}(x) \vee \mathsf{boss}(x, f_1(x))$ and $\neg\mathsf{Manag}(x) \vee \mathsf{Superv}(x) \vee \mathsf{Manag}(f_1(x))$. These will then be transformed by Ξ to the clauses $\neg\mathsf{Manag}(x) \vee \mathsf{Superv}(x)$, $\neg\mathsf{Manag}(x) \vee \mathsf{boss}(x, f_1(x))$ and $\neg\mathsf{Manag}(x) \vee \mathsf{Manag}(f_1(x))$. Finally, Ψ will produce the following datalog rules.

$$\mathsf{Manag}(x) \to \mathsf{Superv}(x) \qquad (U_5^1)$$
$$\mathsf{Manag}(x) \to \mathsf{boss}(x, c_1) \qquad (U_5^2)$$
$$\mathsf{Manag}(x) \to \mathsf{Manag}(c_1) \qquad (U_5^3)$$

Rule T_6 will be transformed by Σ into the clause $\neg\mathsf{Manag}(x) \vee \mathsf{boss}(x, f_2(x))$, which in turn will be transformed by Ψ into the datalog rule U_6.

$$\mathsf{Manag}(x) \to \mathsf{boss}(x, c_2) \qquad (U_6)$$

Hence, $\mathcal{U}(\mathcal{K})$ comprises the facts D_1–D_7 and the datalog rules T_1–T_3, U_4, U_5^1–U_5^3, and U_6. One can easily verify that $\mathsf{cert}(q^{ex}, \mathcal{U}(\mathcal{K}^{ex})) = \{Sue, Dan, Rob, Jo\}$.

The following lemma captures the properties of these transformations.

Proposition 1. *Let $\mathcal{K}$ be a knowledge base and q be a query. Then:*

1. *$\mathcal{K}$ unsatisfiable $\Leftrightarrow$ $\Sigma(\mathcal{K})$ unsatisfiable $\Rightarrow$ $\Xi(\Sigma(\mathcal{K})) \models \bot$ $\Rightarrow$ $\mathcal{U}(\mathcal{K}) \models \bot$;*
2. *$\mathcal{K}$ satisf. $\Rightarrow$ $\mathsf{cert}(q, \mathcal{K}) = \mathsf{cert}(q, \Sigma(\mathcal{K})) \subseteq \mathsf{cert}(q, \Xi(\Sigma(\mathcal{K}))) \subseteq \mathsf{cert}(q, \mathcal{U}(\mathcal{K}))$.*

5 Computing Lower Bounds

A direct way to compute lower bound query answers given $\mathcal{K}$ and q is to select the datalog fragment $\mathcal{L}(\mathcal{K})$ of $\mathcal{K}$, check its satisfiability, and compute $\mathsf{cert}(q, \mathcal{L}(\mathcal{K}))$ using a datalog engine. By monotonicity of first-order logic, $\mathcal{K}$ entails $\mathcal{L}(\mathcal{K})$, and hence $\mathsf{cert}(q, \mathcal{K})$ is guaranteed to subsume $\mathsf{cert}(q, \mathcal{L}(\mathcal{K}))$. In our running example, the lower bound knowledge base $\mathcal{L}(\mathcal{K}^{ex})$ comprises the facts D_1–D_7 and the datalog rules T_1–T_4, and it can be easily verified that $\mathsf{cert}(q^{ex}, \mathcal{L}(\mathcal{K}^{ex})) = \{Dan\}$.

To improve this bound, we adopt the *combined approach* introduced to handle query answering in $\mathcal{ELHO}_\bot^r$ [20, 12]. Given an $\mathcal{ELHO}_\bot^r$ knowledge base $\mathcal{K}'$ and a query q, the combined approach first exploits the upper bound datalog program $\mathcal{U}(\mathcal{K}')$ to check satisfiability of $\mathcal{K}'$ and to compute $\mathsf{cert}(q, \mathcal{U}(\mathcal{K}'))$. A subsequent filtering step Φ, which is efficiently implementable, guarantees to eliminate all spurious tuples; the resulting answer $\Phi(\mathsf{cert}(q, \mathcal{U}(\mathcal{K}')))$ is thus sound and complete w.r.t. q and $\mathcal{K}'$.

The combined approach is clearly compatible with ours. Given an OWL 2 knowledge base $\mathcal{K}$ and query q, we proceed as follows. First, we select the datalog fragment $\mathcal{K}_1 = \mathcal{L}(\mathcal{K})$, and compute the materialisation $Mat(\mathcal{K}_1)$ using the datalog engine. Second, we select the subset $\mathcal{K}_2$ of $\mathcal{K}$ corresponding to $\mathcal{ELHO}_\bot^r$ axioms and Skolemise existential quantifiers to constants to obtain $\mathcal{U}(\mathcal{K}_2)$. Then, we further compute the answers $\mathsf{cert}(q, \mathcal{U}(\mathcal{K}_2) \cup Mat(\mathcal{K}_1))$. Finally, we apply the filtering step Φ to obtain the final set of lower bound answers. The $\mathcal{ELHO}_\bot^r$ fragment for our running example $\mathcal{K}^{ex}$ consists of rules T_1–T_4 and T_6, and the resulting new lower bound answer of q^{ex} is the set $\{Dan, Rob\}$.

Table 1. SLD proofs of $\perp$ and $q^{ex}(Jo)$ in $\mathcal{U}(\mathcal{K}^{ex})$

$\perp$		$b(J,y) \wedge W(y)$	
$T(x) \wedge b(x,y) \wedge M(y)$	by U_4	$M(J) \wedge W(c_2)$	by U_6
$b(J,y) \wedge M(y)$	by D_6	$W(c_2)$	by D_5
$M(J) \wedge M(c_1)$	by U_5^2	$S(x) \wedge b(x,c_2)$	by T_3
$M(c_1)$	by D_5	$M(x) \wedge b(x,c_2)$	by U_5^1
$M(x)$	by U_3^5	$b(J,c_2)$	by D_5
$\square$	by D_5	$M(J)$	by U_6
		$\square$	by D_5

6 Computing Relevant Fragments

Fragment Definition and Formal Properties The relevant fragments $\mathcal{K}_\perp$ and $\mathcal{K}_{[q,G]}$ are defined in terms of SLD proofs in $\mathcal{U}(\mathcal{K})$. In particular, $\mathcal{K}_\perp$ is defined in terms of proofs for the nullary predicate $\perp$, and $\mathcal{K}_{[q,G]}$ is defined in terms of proofs for each answer in G.

Definition 2. *Let $\mathcal{K}$ be a knowledge base, $q(\mathbf{x})$ be a query, and S be a set of tuples. Then $\mathcal{K}_\perp$ (resp. $\mathcal{K}_{[q,S]}$) is the set of all $\alpha \in \mathcal{K}$ for which there exists $\beta \in \mathcal{U}(\alpha)$ involved in an SLD proof of $\perp$ (resp. $Q(\mathbf{a})$, for some $\mathbf{a} \in S$) in $\mathcal{U}(\mathcal{K})$.*

The properties of these fragments needed to ensure the correctness of our algorithm in Section 3 are summarised in the following theorem.

Theorem 1. *Let $\mathcal{K}$ be a knowledge base, $q(\mathbf{x})$ a conjunctive query, and S a set of tuples. Then,* (i) *$\mathcal{K}$ is satisfiable iff $\mathcal{K}_\perp$ is satisfiable; and* (ii) *if $\mathcal{K}$ is satisfiable, then $\mathcal{K} \models q(\mathbf{a})$ iff $\mathcal{K}_{[q,S]} \cup \mathcal{K}_\perp \models q(\mathbf{a})$ for every $\mathbf{a} \in S$.*

As expected, $\mathcal{K}_\perp$ can be used to determine satisfiability of $\mathcal{K}$. In case $\mathcal{K}$ is found satisfiable, the union of $\mathcal{K}_{[q,G]}$ and $\mathcal{K}_\perp$ can then be used to check the validity of each candidate answer in G (in this case, $\mathcal{K}_\perp$ is still needed to account for the possible interactions between non-Horn rules and rules with empty heads).

Table 1 specifies proofs of $\perp$ and $q^{ex}(Jo)$ in $\mathcal{U}(\mathcal{K}^{ex})$, where predicates and constants are abbreviated to their first letters. By Definition 2, $\mathcal{K}_\perp \cup \mathcal{K}_{[q^{ex},\{Jo\}]}$ subsumes $\{T_3, \dots, T_6, D_5, D_6\}$, and, hence, it entails $q^{ex}(Jo)$, as expected. Note that $\mathcal{K}_{[q,\{Jo\}]}$ alone is not sufficient to show $q^{ex}(Jo)$ since every fragment of $\mathcal{K}^{ex}$ that entails $q^{ex}(Jo)$ must include rule T_4. According to Definition 1, $\mathcal{K}_{[q,\{Jo\}]}$ will include T_4 if and only if U_4 is used in an SLD proof of $q^{ex}(Jo)$ in $\mathcal{U}(\mathcal{K}^{ex})$; however, no such proof will involve U_4 since the goal $q^{ex}(Jo)$ does not involve $\perp$, and there is no way of eliminating $\perp$ from a goal using the rules in $\mathcal{U}(\mathcal{K}^{ex})$ as they do not contain $\perp$ in their bodies.

The proof of Theorem 1 is involved, and details are deferred to the appendix. Nonetheless, we next sketch the arguments behind the proof. A first observation is that, w.l.o.g. we can restrict ourselves to the case where $q(\boldsymbol{x})$ is atomic.

Lemma 1. *Let $\mathcal{K}$ be a knowledge base, $q(\mathbf{x}) = \exists \mathbf{y}\, \varphi(\mathbf{x}, \mathbf{y})$ be a CQ, S be a set of tuples, Q be a fresh predicate, and let $\mathcal{K}' = \mathcal{K}_{[q,S]} \cup \mathcal{K}_\perp$. Then, $\mathcal{K}' \models q(\mathbf{a})$ iff $\mathcal{K}' \cup \{\varphi(\mathbf{x}, \mathbf{y}) \to Q(\mathbf{x})\} \models Q(\mathbf{a})$.*

The crux of the proof relies on the following properties of Ξ (the step in the definition of $\mathcal{U}$ which splits each non-Horn clause C into Horn clauses).

Lemma 2. *Let $\mathcal{N}$ be a set of first-order clauses. Then:*

- *if $C \in \mathcal{N}$ participates in a refutation in $\mathcal{N}$, then every $C' \in \Xi(C)$ is part of an SLD proof of $\bot$ in $\Xi(\mathcal{N})$;*
- *if $C \in \mathcal{N}$ participates in a resolution proof in $\mathcal{N}$ of an atomic query $Q(\mathbf{a})$, then each $C' \in \Xi(C)$ participates in an SLD proof of $\bot$ or $Q(\mathbf{a})$ in $\Xi(\mathcal{N})$.*

Thus, by Lemma 2, each resolution proof in a set of clauses $\mathcal{N}$ can be mapped to SLD proofs in $\Xi(\mathcal{N})$ that "preserves" the participating clauses. The following lemma allows us to restate Lemma 2 for $\Psi \circ \Xi$ instead of Ξ.

Lemma 3. *Let $\mathcal{H}$ be a set of first-order Horn clauses, $Q(\mathbf{x})$ be an atomic query, and $\mathbf{a}$ be a tuple of constants. If a clause C participates in an SLD proof of $Q(\mathbf{a})$ in $\mathcal{H}$, then $\Psi(C)$ participates in an SLD proof of $Q(\mathbf{a})$ in $\Psi(\mathcal{H})$.*

With these Lemmas, we can exploit refutational completeness of resolution and the entailment preservation properties of Skolemisation to show Theorem 1.

Fragment Computation The computation of the relevant fragments requires a scalable algorithm for "tracking" all rules and facts involved in SLD proofs for datalog programs. We next present a novel technique that delegates this task to the datalog engine itself. The main idea is to extend the datalog program with additional rules that are responsible for the tracking; in this way, the relevant rules and facts can be obtained from the materialisation of the modified program.

Definition 3. *Let $\mathcal{K}$ be a datalog KB and let F be a set of facts in $Mat(\mathcal{K})$. Then, $\Delta(\mathcal{K}, F)$ is the datalog program containing the rules and facts given next:*

- *each rule and fact in $\mathcal{K}$;*
- *a fact $\bar{P}(\mathbf{a})$ for each fact $P(\boldsymbol{a})$ in F;*
- *the following rules for each $r \in \mathcal{K}$ of the form $B_1(\mathbf{x}_1), \ldots, B_m(\mathbf{x}_m) \to H(\mathbf{x})$, and $1 \leq i \leq m$, with $\mathbf{c_r}$ a fresh constant for each r, and $\mathbf{S}$ a fresh predicate:*

$$\bar{H}(\mathbf{x}) \wedge B_1(\mathbf{x}_1) \wedge \ldots, B_m(\mathbf{x}_m) \to \mathbf{S}(\mathbf{c_r}) \tag{1}$$

$$\bar{H}(\mathbf{x}) \wedge B_1(\mathbf{x}_1), \ldots \wedge B_m(\mathbf{x}_m) \to \bar{B}_i(\mathbf{x}_i) \tag{2}$$

The auxiliary predicates $\bar{P}$ are used to record facts involved in proofs; in particular, if $\bar{P}(\boldsymbol{c})$ is contained in $Mat(\Delta(\mathcal{K}, F))$, we can conclude that $P(\boldsymbol{c})$ participates in an SLD proof in $\mathcal{K}$ of a fact in F. Furthermore, each rule $r \in \mathcal{K}$ is represented by a fresh constant $\mathbf{c_r}$, and $\mathbf{S}$ is a fresh predicate that is used to record rules of $\mathcal{K}$ involved in proofs. In particular, if $\mathbf{S}(\mathbf{c_r})$ is contained in $Mat(\Delta(\mathcal{K}, F))$, we can conclude that rule r participates in an SLD proof in $\mathcal{K}$ of a fact in F. The additional rules (1) and (2) are responsible for the tracking and make sure that the materialisation of $\Delta(\mathcal{K}, F)$ contains the required information. Indeed, if there is an instantiation $B_1(\mathbf{a}_1) \wedge \ldots \wedge B_m(\mathbf{a}_m) \to H(\mathbf{a})$ of a rule $r \in \Delta$, then, by virtue of (1), $\mathbf{c_r}$ will be added to $\mathbf{S}$, and, by virtue of (2), each $\bar{B}_i(\mathbf{a}_i)$, for $1 \leq i \leq m$, will be derived. Correctness is established as follows.

Theorem 2. *Let $\mathcal{K}$ be a datalog knowledge base and let F be a set of facts in $Mat(\mathcal{K})$. Then, a fact $P(\mathbf{a})$ (resp. a rule r) in $\mathcal{K}$ participates in an SLD proof of some fact in F iff $\bar{P}(\mathbf{a})$ (resp. $\mathbf{S}(\mathbf{c_r})$) is in $Mat(\Delta(\mathcal{K}, F))$.*

7 Summarisation and Answer Dependencies

Once the relevant fragment has been computed, we check, using the fully-fledged reasoner, whether each candidate answer is entailed. This can be computationally expensive if the fragment is large, or there are many candidate answers to verify. To address these issues, we exploit summarisation techniques [4] to efficiently prune candidate answers. The idea behind summarisation is to "shrink" the data by merging constants instantiating the same unary predicates. Since summarisation is equivalent to extending the knowledge base with equality assertions, the summarised knowledge base entails the original one by monotonicity.

Definition 4. *Let $\mathcal{K}$ be a knowledge base. A* type *T is a set of unary predicates; for a constant a in $\mathcal{K}$, we say that $T = \{A \mid A(a) \in \mathcal{K}\}$ is the type for a. Furthermore, for each type T, let c_T be a globally fresh constant uniquely associated with T. The* summary function *over $\mathcal{K}$ is the substitution σ mapping each constant a in $\mathcal{K}$ to c_T, where T is the type for a. Finally, the knowledge base $\sigma(\mathcal{K})$ obtained by replacing each constant a in $\mathcal{K}$ with $\sigma(a)$ is called the* summary *of $\mathcal{K}$.*

By summarising a knowledge base, we overestimate query answers [4].

Proposition 2. *Let $\mathcal{K}$ be a knowledge base, and let σ be the summary function over $\mathcal{K}$. Then, for every query q we have $\sigma(\mathsf{cert}(q, \mathcal{K})) \subseteq \mathsf{cert}(\sigma(q), \sigma(\mathcal{K}))$.*

Summarisation can be exploited to detect spurious answers in G: if a tuple is not in $\mathsf{cert}(\sigma(q), \sigma(\mathcal{K}))$, then it is not in $\mathsf{cert}(q, \mathcal{K})$. Since summarisation can significantly reduce the size of a knowledge base, we can efficiently detect non-answers even if checking them over the summary requires calling the OWL reasoner.

Corollary 1. *Let $\mathcal{K}$ be a knowledge base, let q be a query, let S be a set of tuples, and let $\mathcal{K}' = \mathcal{K}_{[q,S]} \cup \mathcal{K}_{\perp}$. Furthermore, let σ be the summary function over $\mathcal{K}'$. Then, $\sigma(\mathcal{K}') \not\models \sigma(q(\mathbf{a}))$ implies $\mathcal{K} \not\models q(\mathbf{a})$ for each $\mathbf{a} \in S$.*

Finally, we try to further reduce the calls to the fully-fledged reasoner by exploiting dependencies between the candidate answers. Consider tuples $\mathbf{a}$ and $\mathbf{b}$ in G and the dataset $\mathcal{D}$ in the fragment $\mathcal{K}_{[q,G]} \cup \mathcal{K}_{\perp}$; furthermore, suppose we can find an endomorphism h of $\mathcal{D}$ in which $h(\mathbf{a}) = \mathbf{b}$. If we can determine (by calling the fully-fledged reasoner) that $\mathbf{b}$ is a spurious answer, then so must be $\mathbf{a}$; as a result, we no longer call the reasoner to check $\mathbf{a}$. We exploit this idea to compute a dependency graph having candidate answers as nodes and an edge $(\mathbf{a}, \mathbf{b})$ whenever an endomorphism in $\mathcal{D}$ exists mapping $\mathbf{a}$ to $\mathbf{b}$. Computing endomorphisms is computationally hard, so we have implemented a sound (but incomplete) greedy algorithm that approximates the dependency graph.

Data	DL	Axioms	Facts
LUBM(n)	$\mathcal{SHI}$	93	$10^5 n$
UOBM$^-$(n)	$\mathcal{SHIN}$	314	$2 \times 10^5 n$
FLY	$\mathcal{SRI}$	144,407	6,308
DBPedia$^+$	$\mathcal{SHOIN}$	1,757	12,119,662
NPD	$\mathcal{SHIF}$	819	3,817,079

Table 2. Statistics for test data

Strategy	Solved	|Univ|	$t_{\text{avg.}}$
RL Bounds	14	1000	18.4
+ EL Lower Bound	22	1000	11.7
+ Sum, Dep	24	100	29.6

Table 3. Result for LUBM

Strategy	Solved	|Univ|	$t_{\text{avg.}}$
RL Bounds	12	500	0.7
+ Summarisation	14	60	14.0
+ Dependencies	15	1	1.8

Table 4. Result for UOBM$^-$

8 Evaluation

We have implemented a prototype system, called PAGOdA, based on RDFox and HermiT (v. 1.3.8). For testing, we used the LUBM and UOBM benchmarks, as well as the Fly Anatomy ontology, DBPedia and NPD FactPages; their key features are summarised in Table 2. Our system, test data, ontologies, and queries are available online.[2] We compared our system with Pellet (v. 2.3.1) and TrOWL [21] on all datasets. While Pellet is sound and complete, TrOWL relies on approximate reasoning and does not provide correctness guarantees. Tests were performed on a 16 core 3.30GHz Intel Xeon E5-2643 with 125GB of RAM, and running Linux 2.6.32. For each test, we measured materialisation times for upper and lower bound, the time to answer each query, and the number of queries that can be fully answered using different techniques. All times are in seconds.

Materialisation is fast on LUBM [8]: it takes 319s (341s) to materialise the basic lower (upper) bound entailments for LUBM(1000). These bounds match for all 14 standard LUBM queries, and we have used 10 additional queries for which this is not the case; we tested our system on all 24 queries (see Table 3 for a summary of the results). The refined lower bound was materialised in 366s, and it matches the upper bound for 8 of the 10 additional queries; thus, our system could answer 22 of the 24 queries over LUBM(1000) efficiently in 12s on average.[3] For the remaining 2 queries, we could scale to LUBM(100) in reasonable time. On LUBM(100) the gaps contain 29 and 14 tuples respectively, none of which were eliminated by summarisation; however, exploiting dependencies between gap tuples reduced the calls to HermiT to only 3 and 1 respectively, with the majority of time taken in extraction (avg. 45s) and HermiT calls (avg. 281s). On LUBM(1000), Pellet ran out of memory. For LUBM(100), Pellet took on average 8.2s to answer the standard queries with an initialisation overhead of 388s. TrOWL timed out after 1h on LUBM(100).

[2] http://www.cs.ox.ac.uk/isg/tools/PAGOdA/

[3] Average query answering times are measured after materialisation.

UOBM is an extension of LUBM [26]. Query answering over UOBM requires equality reasoning (e.g., to deal with cardinality constraints), which is not natively supported by RDFox,[4] so we have used a slightly weakened ontology UOBM$^-$ for which equality is not required. Materialisation is still fast on UOBM$^-$(500): it takes 346s (378s) to materialise the basic lower (upper) bound entailments. We have tested the 15 standard queries (see Table 4). The basic lower and upper bounds match for 12 queries; our system is efficient for these queries, with an average query answering time of less than 1s over UOBM$^-$(500). For 2 of the remaining queries, summarisation prunes all candidate answers. Average times for these queries were under 15s for UOBM$^-$(60). For the one remaining query, summarisation rules out 6245 among 6509 answers in the gap, and the dependency analysis groups all the remaining individuals. HermiT, however, takes 20s to check the representative answer for UOBM$^-$(1), and 4000s for UOBM$^-$(10). Pellet times out even on UOBM$^-$(1). TrOWL took 237s on average to answer 14 out of the 15 queries over UOBM$^-$(60).[5] Furthermore, a comparison with our system reveals that TrOWL answers may be neither sound nor complete for most test queries.

Fly Anatomy is a complex ontology, rich in existential axioms, and including a dataset with over 6,000 facts. We tested it with five queries provided by the developers the ontology. It took 88s (106s) to materialise lower (upper) bound entailments. The basic lower bounds for all queries are empty, whereas the refined lower bounds (which take 185s to materialise) match with the upper bound in all cases; as a result, we can answer the queries in 0.2s on average. Pellet fails to answer queries given a 1h timeout, and TrOWL returns only empty answers.

In contrast to Fly, the DBPedia dataset is relatively large, but the ontology is simple. To provide a more challenging test, we have used the LogMap ontology matching system [10] to extend DBPedia with the tourism ontology which contains both disjunctive and existential axioms. Since the tested systems report errors on datatypes, we have removed all axioms and facts involving datatypes. It takes 45s (47s) to materialise the basic lower (upper) bound entailments. The upper bound was unsatisfiable and it took 2.6s to check satisfiability of the $\mathcal{K}_\bot$ fragment. We queried for instances of all 441 atomic concepts. Bounds matched in 439 cases (using the refined lower bound), and these queries were answered in 0.3s on average. Summarisation filtered out all gap tuples for the remaining two queries. The answer time for both queries was less than 3s. Pellet takes 280.9s to initialise and answers each query in an average time of 16.2s. TrOWL times out after 1h.

The NPD FactPages ontology describes petroleum activities on the Norwegian continental shelf. The ontology is not Horn, and it includes existential axioms. As in the case of DBPedia, we removed axioms involving datatypes. Its dataset has about 4 million triples; it takes 17s (22s) to materialise the lower (upper) bound entailments. The upper bound is unsatisfiable, and it took 30s to check satisfiability of $\mathcal{K}_\bot$. We queried for the instances of the 329 atomic

[4] RDFox supports equality via its axiomatisation as a congruence relation.
[5] An exception is reported for the remaining query.

concepts, and could answer all queries in 2.5s on average. Queries with matching bounds (294 out of 329) could be answered on 0.1s, and average query answering time was 3s. TrOWL took 1.3s to answer queries on average; answers were complete for 320 out of the 329 queries.

9 Related Techniques

The Screech system [22] exploits the KAON2 reasoner [9] to rewrite a $\mathcal{SHIQ}$ ontology into disjunctive datalog while preserving atomic queries, and then transforms $\vee$ into $\wedge$; the resulting over-approximation can be used to compute upper bound query answers. This technique is restricted to $\mathcal{SHIQ}$ ontologies and atomic queries; furthermore, the set of rules obtained from KAON2 can be expensive to compute, as well as of exponential size. Both the Quill system [18] and the work of [26] under-approximate the ontology into OWL 2 QL; however, neither approximation is independent of both query and data, and using OWL 2 QL increases the chances that the approximated ontology will be unsatisfiable.

The SHER system uses summarisation to efficiently compute an upper bound answer, with exact answers then being computed via successive relaxations [3, 4]. The technique has been shown to be scalable, but it is only known to be applicable to $\mathcal{SHIN}$ and atomic queries, and is less modular than our approach. In contrast, our approach can profitably exploit the summarisation technique, and could even improve scalability for the hardest queries by replacing HermiT with SHER when the extracted fragment is $\mathcal{SHIN}$.

10 Discussion

We have proposed a novel approach for query answering that integrates scalable and complete reasoners to provide pay-as-you-go performance. Our evaluation shows that 772 of the 814 test queries could be answered using highly scalable lower and upper bound computations, 39 of the remaining 42 queries yielded to extraction and summarisation techniques, and even for the remaining 3 queries our fragment extraction and dependency techniques greatly improved scalability. Our approach is complementary to other optimisation efforts, and could immediately benefit from alternative techniques for efficiently computing lower bounds and/or a more efficient OWL reasoner. Our technical results are very general, and hold for any language L captured by generalised rules.

There are still many possibilities for future work. For the immediate future, our main focus will be improving the fragment extraction and checking techniques so as to improve scalability for the hardest queries.

Acknowledgements. This work was supported by the Royal Society, the EPSRC projects Score!, Exoda, and MaSI^3, and the FP7 project OPTIQUE.

References

1. Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., Velkov, R.: OWLIM: A family of scalable semantic repositories. Semantic Web 2(1), 33–42 (2011)
2. Cuenca Grau, B., Motik, B., Stoilos, G., Horrocks, I.: Completeness guarantees for incomplete ontology reasoners: Theory and practice. J. Artif. Intell. Res. (JAIR) 43, 419–476 (2012)
3. Dolby, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Schonberg, E., Srinivas, K., Ma, L.: Scalable semantic retrieval through summarization and refinement. In: AAAI. pp. 299–304 (2007)
4. Dolby, J., Fokoue, A., Kalyanpur, A., Schonberg, E., Srinivas, K.: Scalable highly expressive reasoner (SHER). J. Web Sem. 7(4), 357–361 (2009)
5. Eiter, T., Ortiz, M., Simkus, M.: Conjunctive query answering in the description logic $\mathcal{SH}$ using knots. J. Comput. Syst. Sci. 78(1), 47–85 (2012)
6. Gallier, J.H.: Logic for Computer Science: Foundations of Automatic Theorem Proving. Wiley (1987)
7. Glimm, B., Lutz, C., Horrocks, I., Sattler, U.: Conjunctive query answering for the description logic $\mathcal{SHIQ}$. J. Artif. Intell. Res. (JAIR) 31, 157–204 (2008)
8. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. J. Web Sem. 3(2-3), 158–182 (2005)
9. Hustadt, U., Motik, B., Sattler, U.: Reasoning in description logics by a reduction to disjunctive datalog. J. Autom. Reasoning 39(3), 351–384 (2007)
10. Jiménez-Ruiz, E., Cuenca Grau, B., Zhou, Y., Horrocks, I.: Large-scale interactive ontology matching: Algorithms and implementation. In: ECAI. pp. 444–449 (2012)
11. Kollia, I., Glimm, B.: Optimizing SPARQL query answering over OWL ontologies. J. Artif. Intell. Res. (JAIR) 48, 253–303 (2013)
12. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to ontology-based data access. In: IJCAI. pp. 2656–2661 (2011)
13. Möller, R., Neuenstadt, C., Özçep, Ö.L., Wandelt, S.: Advances in accessing big data with expressive ontologies. In: Description Logics. pp. 842–853 (2013)
14. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language Profiles. W3C Recommendation (27 October 2009), available at `http://www.w3.org/TR/owl2-profiles/`
15. Motik, B., Nenov, Y., Piro, R., Horrocks, I., Olteanu, D.: Parallel materialisation of datalog programs in main-memory rdf databases. In: AAAI (2014)
16. Motik, B., Patel-Schneider, P.F., Parsia, B.: OWL 2 Web Ontology Language Structural Specification and Functional-style Syntax. W3C Recommendation (27 October 2009 2009), available at `http://www.w3.org/TR/owl2-syntax/`
17. Motik, B., Shearer, R., Horrocks, I.: Hypertableau reasoning for description logics. J. Artif. Intell. Res. (JAIR) 36, 165–228 (2009)
18. Pan, J.Z., Thomas, E., Zhao, Y.: Completeness guaranteed approximations for OWL-DL query answering. In: Description Logics (2009)
19. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. Web Sem. 5(2), 51–53 (2007)
20. Stefanoni, G., Motik, B., Horrocks, I.: Introducing nominals to the combined query answering approaches for $\mathcal{EL}$. In: AAAI (2013)
21. Thomas, E., Pan, J.Z., Ren, Y.: TrOWL: Tractable OWL 2 reasoning infrastructure. In: ESWC (2). pp. 431–435 (2010)

22. Tserendorj, T., Rudolph, S., Krötzsch, M., Hitzler, P.: Approximate OWL-reasoning with Screech. In: RR. pp. 165–180 (2008)
23. Urbani, J., van Harmelen, F., Schlobach, S., Bal, H.E.: QueryPIE: Backward reasoning for OWL Horst over very large knowledge bases. In: International Semantic Web Conference (1). pp. 730–745 (2011)
24. Urbani, J., Kotoulas, S., Maassen, J., van Harmelen, F., Bal, H.E.: WebPIE: A web-scale parallel inference engine using MapReduce. J. Web Sem. 10, 59–75 (2012)
25. W3C SPARQL Working Group: SPARQL 1.1 Overview. W3C Recommendation (21 March 2013), available at `http://www.w3.org/TR/sparql11-overview/`
26. Wandelt, S., Möller, R., Wessel, M.: Towards scalable instance retrieval over ontologies. Int. J. Software and Informatics 4(3), 201–218 (2010)
27. Wu, Z., Eadon, G., Das, S., Chong, E.I., Kolovski, V., Annamalai, M., Srinivasan, J.: Implementing an inference engine for RDFS/OWL constructs and user-defined rules in oracle. In: ICDE. pp. 1239–1248 (2008)
28. Zhou, Y., Cuenca Grau, B., Horrocks, I., Wu, Z., Banerjee, J.: Making the most of your triple store: query answering in OWL 2 using an RL reasoner. In: WWW. pp. 1569–1580 (2013)
29. Zhou, Y., Nenov, Y., Cuenca Grau, B., Horrocks, I.: Complete query answering over Horn ontologies using a triple store. In: ISWC (1). pp. 720–736 (2013)
30. Zhou, Y., Nenov, Y., Cuenca Grau, B., Horrocks, I.: Pay-as-you-go OWL query answering using a triple store. In: AAAI (2014)

A Correctness of Fragment Extraction

We will now show that the relevant subsets extraction is in fact sound and complete for any OWL 2 ontologies by proving that Theorem 1 is correct. We first look at several definitions that we will used later on.

Definition 5. *With* B *we denote a resolution calculus, consisting of the following rules:*

$$\frac{C \vee A \quad D \vee \neg B}{C\sigma \vee D\sigma} \qquad (\textit{Resolution})$$

where $\sigma = \mathsf{MGU}(A, B)$*, the clauses* $C \vee A$ *and* $D \vee \neg B$ *are called the premises, and* $C\sigma \vee D\sigma$ *is the called the conclusion of the resolution step.*

$$\frac{C \vee A \vee B}{C\sigma \vee A\sigma} \qquad (\textit{Factoring})$$

where $\sigma = \mathsf{MGU}(A, B)$ *and the clause* $C\sigma \vee A\sigma$ *is called the conclusion of the factoring step.*

Definition 6. *Let* Γ *be a set of first-order clauses. A* resolution derivation *of a goal clause* G *in* Γ *is a sequence of clauses* $H_1, \ldots, H_l$ *that satisfies the following conditions.*

1. $H_l = G$.
2. *For each* $1 \leq i \leq l$, H_i *is the conclusion of an inference by* B *from premises in* $\Gamma \cup \{H_1, \ldots, H_{i-1}\}$*. If a premise is* $H_p \in \{H_1, \ldots, H_{i-1}\}$, H_i *is called a* successor *of* H_p.
3. *Every* H_i *has at least one successor in the derivation for each* $i < l$.

Particularly, a refutation *is a derivation with the empty clause in the end. The* support *of a derivation* $\mathsf{sup}(G_1, \ldots, G_l)$ *is defined as the set of clauses in* Γ *that have been used to derive a clause* G_i*, for some* $1 \leq i \leq l$.

Definition 7. *Let* Γ *be a set of first-order clauses, an* SLD proof *of a* goal G *in* Γ *is a sequence of clauses*

$$G_0 = \neg G \overset{C_1}{\rightsquigarrow} G_1 \rightsquigarrow \ldots \rightsquigarrow G_{l-1} \overset{C_l}{\rightsquigarrow} G_l$$

where G_l *is the empty clause and* $C_i \in \Gamma$ *and each* G_{i+1} *is the conclusion of a resolution step from* G_i *and* C_{i+1}.

Obviously, there is a one to one correspondence between the SLD proofs and the SLD proofs defined in the preliminaries, where C_i is from the clauses that corresponds to the datalog rules and each goal clause here is a negation of a goal there. We will use the new definition in the appendix.

From Lemma 1 it follows that we only need to show theorem in case of atomic queries. So we restrict ourselves to atomic queries from now on.

Lemma 3. *Let $\mathcal{H}$ be a set of first-order Horn clauses, $Q(\mathbf{x})$ be an atomic query, and $\mathbf{a}$ be a tuple of constants. If a clause C participates in an SLD proof of $Q(\mathbf{a})$ in $\mathcal{H}$, then $\Psi(C)$ participates in an SLD proof of $Q(\mathbf{a})$ in $\Psi(\mathcal{H})$.*

Proof. If there is a clause C participates in an SLD proof of $Q(\mathbf{a})$ in $\mathcal{H}$, we assume that the SLD proof to be the sequence (3) where $C = C_i$ for some $1 \leq i \leq l$ and the clause $C_k \in \mathcal{H}$ for each k.

$$(G_0 = \neg Q(\mathbf{a})) \overset{C_1}{\rightsquigarrow} G_1 \rightsquigarrow \ldots \rightsquigarrow (G_l = \Box) \tag{3}$$

Given the SLD-proof (3), we construct the sequence inductively on the index i

$$(P_0 = \neg Q(\mathbf{a})) \overset{\Psi(C_1)}{\rightsquigarrow} P_1 \rightsquigarrow \ldots \rightsquigarrow P_l \tag{4}$$

such that it satisfies the following properties:

Property 1. P_i is the resolvent of P_{i-1} and $\Psi(C_i)$ for each $1 \leq i \leq l$.
Property 2. $\Psi(G_i) = P_i\sigma_i$ for a substitution σ_i for each $0 \leq i \leq l$.

If both of the above properties hold for each i, then the sequence (4) is an SLD-proof of $Q(\mathbf{a})$ in $\Psi(\mathcal{H})$, and thus, the lemma holds.

- Base case: $i = 0$, $P_i = \neg Q(\mathbf{a}) = G_i$ and then σ_0 can be the identical substitution;
- Inductive case: Assume the following conditions.

$$G_{i-1} = \neg A_1^G \vee \ldots \vee \neg A_n^G \tag{5}$$
$$C_i = \neg B_1 \vee \ldots \vee \neg B_m \vee H \tag{6}$$
$$\theta_i = \mathsf{MGU}(A_k^G, H) \tag{7}$$
$$P_{i-1} = \neg A_1^P \vee \neg \ldots \vee \neg A_n^P \tag{8}$$

  Then G_i is the following clause.

$$(\neg A_1^G \vee \ldots \vee \neg A_{k-1}^G \vee \neg B_1 \vee \ldots \vee \neg B_m \vee \neg A_{k+1}^G \vee \ldots A_n^G)\theta_i$$

  - We here prove that P_{i-1} and $\Psi(C_i)$ are resolvable.

    According to the induction hypothesis, we have the fact $\Psi(A_j^G) = A_j^P\sigma_{i-1}$ for each $1 \leq j \leq n$. Let

$$X_1 = \{x \mid x \text{ appears in } A_k^P\}, \quad Y = \{x \mid x \text{ appears in } A_j^P \text{ for a } j\} \setminus X_1,$$
$$X_2 = \{x \mid x \text{ appears in } H\}, \quad Z = \{x \mid x \text{ appears in a } B_j\} \setminus X_2.$$

    We can assume that X_1, X_2, Y, Z are pair-wise disjoint, otherwise the condition can be satisfied by renaming variables; and further assume that $\mathsf{Vars}(\theta_i)$ — all the variables appear in the domain or image of θ_i is a subset of $X_1 \cup X_2$, otherwise, we can trim or renaming the substitution accordingly.

Since $A_k^G\theta_i = H\theta_i$, it is easy to verify that $\Psi(A_k^G)\Psi(\theta_i) = \Psi(H)\Psi(\theta_i)$ since Ψ replace the function symbols with corresponding constants. So $\Psi(\theta_i)$ is a unifier for $\Psi(A_k^G)$ and $\Psi(H)$, and hence a unifier for $A_k^P\sigma_{i-1}$ and $\Psi(H)$. Since X_1 and X_2 are disjoint, $H = H\sigma_{i-1}$ and then $\sigma_{i-1}\Psi(\theta_i)$ is a unifier of A_k^P and $\Psi(H)$. So P_{i-1} and $\Psi(C_i)$ are resolvable.

- Let $\lambda_i = \mathsf{MGU}(A_k^P, \Psi(H))$. W.l.o.g. we assume that $\mathsf{Vars}(\lambda_i) \subseteq X_1 \cup X_2$. Then P_i is defined as the resolvent of P_{i-1} and $\Psi(C_i)$ of the following form.

$$(\neg A_1^P \vee \ldots \vee \neg A_{k-1}^P \vee \neg\Psi(B_1) \vee \ldots \vee \neg\Psi(B_m) \vee \neg A_{k+1}^P \vee \ldots A_n^P)\lambda_i$$

- Up to now, we have proved that we can construct P_i as the resolvent of P_{i-1} and $\Psi(C_i)$. We next prove that there exists a substitution σ_i s.t. $\Psi(G_i) = P_i\sigma_i$. Since $\lambda_i = \mathsf{MGU}(A_k^P, \Psi(H))$ and $\sigma_{i-1}\Psi(\theta_i)$ is a unifier of A_k^P and $\Psi(H)$, then there exists a substitution σ' s.t.

$$A_k^P\lambda_i\sigma' = A_k^P\sigma_{i-1}\Psi(\theta_i), \tag{9}$$
$$\Psi(H)\lambda_i\sigma' = \Psi(H)\sigma_{i-1}\Psi(\theta_i) \tag{10}$$

$\Psi(G_i)$ is the following clause.

$$(\neg\Psi(A_1^G) \vee \ldots \vee \neg\Psi(A_{k-1}^G) \vee \neg\Psi(B_1)\vee$$
$$\ldots \vee \neg\Psi(B_m) \vee \neg\Psi(A_{k+1}^G) \vee \ldots \Psi(A_n^G))\Psi(\theta_i)$$

We next prove $\Psi(G_i) = P_i\sigma_i$ for the following σ_i.

$$\sigma_i(x) = \begin{cases} \Psi(\theta_i)(\sigma_{i-1}(x)) & x \in Y \\ \text{undefined} & x \in Z \\ \sigma'(x) & \text{otherwise} \end{cases}$$

  * To prove $\Psi(A_j^G)\Psi(\theta_i) = A_j^P\lambda_i\sigma_i$, it suffices to show that $A_j^P\sigma_{i-1}\Psi(\theta_i) = A_j^P\lambda_i\sigma_i$ for each $1 \le j < k$ or $k < j \le n$. We show this by discussing the image of each variable x that appears in A_j^P:
    - If $x \in X_1$, then x appears in A_k^P and, according to condition (9), we have $x\sigma_{i-1}\Psi(\theta_i) = x\lambda_i\sigma_i$
    - If $x \in Y$, $x\sigma_{i-1}\Psi(\theta_i) = x\lambda_i\sigma_i$ according to the definition of σ_i.
  * To prove $\Psi(B_j)\Psi(\theta_i) = \Psi(B_j)\lambda_i\sigma_i$ for each $1 \le j \le m$, we discuss the image of each variable x that appears in B_j:
    - If $x \in X_2$, then x appears in H and, according to condition (10), we have $x\sigma_{i-1}\Psi(\theta_i) = x\lambda_i\sigma_i$
    - If $x \in Z$, both $x\Psi(\theta_i)$ and $x\lambda_i\sigma_i$ are still the variable x itself, because θ_i, λ_i and σ_i are not defined on Z.

Theorem 1. *Let $\mathcal{K}$ be a knowledge base, $q(\mathbf{x})$ a conjunctive query, and S a set of tuples. Then,* (i) *$\mathcal{K}$ is satisfiable iff $\mathcal{K}_\perp$ is satisfiable; and* (ii) *if $\mathcal{K}$ is satisfiable, then $\mathcal{K} \models q(\mathbf{a})$ iff $\mathcal{K}_{[q,S]} \cup \mathcal{K}_\perp \models q(\mathbf{a})$ for every $\mathbf{a} \in S$.*

Proof. The "if" direction for the above two claims is trivial due to monotonicity of first-order logic. Hence, we only need to show the "only if" direction. We then prove the two claims separately.

- If $\mathcal{K}$ is unsatisfiable, then $\mathcal{K}^{\Sigma} = \Sigma(\mathcal{K})$ is also unsatisfiable since Skolemisation is satisfiability-preserving. So there is a refutation in $\mathcal{K}^{\Sigma}$. Let $G_1, \ldots, G_l$ be a refutation in $\mathcal{K}^{\Sigma}$. By applying Lemma 2 and Lemma 3, for each $C \in \mathsf{sup}(G_1, \ldots, G_l)$, we have that every $C' \in \Psi \circ \Xi(C)$ is in the support of an SLD proof of $\bot$ in $\Psi \circ \Xi(\mathcal{K}_{\Sigma})$. So $\mathsf{sup}(G_1, \ldots, G_l) \subseteq \Sigma(\mathcal{K}_{\bot})$ and hence $\Sigma(\mathcal{K}_{\bot})$ is unsatisfiable, which is equisatisfiable with $\mathcal{K}_{\bot}$.
- If $\mathcal{K}$ is satisfiable, then $\mathcal{K}^{\Sigma} = \Sigma(\mathcal{K})$ is satisfiable. For each $\mathbf{a} \in S$ s.t. $\mathcal{K} \models q(\mathbf{a})$, we also have $\mathcal{K}^{\Sigma} \models q(\mathbf{a})$. Then $q(\mathbf{a})$ is derivable by B in $\mathcal{K}^{\Sigma}$. Assume $G_1, \ldots, G_l$ is a resolution derivation of $q(\mathbf{a})$ in $\mathcal{K}_{\Sigma}$. Similarly, by applying Lemma 2 and Lemma 3, for every $C \in \mathsf{sup}(G_1, \ldots, G_l)$, we have that $C' \in \Psi \circ \Xi(C)$ is in the support of an SLD proof of $\bot$ or $q(\mathbf{a})$ in $\Psi \circ \Xi(C)$. So $\mathsf{sup}(G_1, \ldots, G_l) \subseteq \Sigma(\mathcal{K}_{[q,S]} \cup \mathcal{K}_{\bot})$. Then $\Sigma(\mathcal{K}_{[q,S]} \cup \mathcal{K}_{\bot}) \models q(\mathbf{a})$, i.e. $\Sigma(\mathcal{K}_{[q,S]} \cup \mathcal{K}_{\bot} \cup \{\neg q(\mathbf{a})\})$ is unsatisfiable. So $\mathcal{K}_{[q,S]} \cup \mathcal{K}_{\bot} \cup \{\neg q(\mathbf{a})\}$ is unsatisfiable, and hence $\mathcal{K}_{[q,S]} \cup \mathcal{K}_{\bot} \models q(\mathbf{a})$.

In the following proof we will use Lemma 4, which was shown in [6].

Lemma 4. *Let $\mathcal{K}^H$ be a set of first-order Horn clauses. If there is a resolution derivation $N_1, \ldots, N_l$ of a fact G in $\mathcal{K}^H$, then there is an SLD proof of G in $\mathcal{K}^H$ of the form $G_0 = \neg G \overset{C_1}{\rightsquigarrow} G_1 \rightsquigarrow \ldots \rightsquigarrow G_{l-1} \overset{C_l}{\rightsquigarrow} G_l$ s.t. $\mathsf{sup}(N_1, \ldots, N_l) = \{C_1, \ldots, C_l\}$.*

Lemma 2. *Let $\mathcal{N}$ be a set of first-order clauses. Then:*

- *if $C \in \mathcal{N}$ participates in a refutation in $\mathcal{N}$, then every $C' \in \Xi(C)$ is part of an SLD proof of $\bot$ in $\Xi(\mathcal{N})$;*
- *if $C \in \mathcal{N}$ participates in a resolution proof in $\mathcal{N}$ of an atomic query $Q(\mathbf{a})$, then each $C' \in \Xi(C)$ participates in an SLD proof of $\bot$ or $Q(\mathbf{a})$ in $\Xi(\mathcal{N})$.*

Proof. According to Lemma 4, it suffices to prove the following two claims.

($C1$) if $C \in \mathcal{N}$ participates in a refutation in $\mathcal{N}$, then every $C' \in \Xi(C)$ is part of a resolution derivation of $\bot$ in $\Xi(\mathcal{N})$;
($C2$) if $C \in \mathcal{N}$ participates in a resolution proof in $\mathcal{N}$ of an atomic query $Q(\mathbf{a})$, then each $C' \in \Xi(C)$ participates in a resolution derivation of $\bot$ or $Q(\mathbf{a})$ in $\Xi(\mathcal{N})$.

We next prove ($C1$). Let $G_1, \ldots, G_l$ be a refutation in $\mathcal{N}$. We construct $P_1, \ldots, P_l$ a sequence of sets of clauses as follows.

- If G_i is the conclusion of a resolution step from two clauses C, D with the substitution σ on the literal L in C and the literal $\neg L$ in D, then we define $S_i(X)$ as follows for $X = C$ or $X = D$.

$$S_i(X) \triangleq \begin{cases} \Xi(X) & X \in \mathcal{N} \\ P_j & X = G_j \text{ for some } 1 \leq j < i \end{cases}$$

Assume that we have $S_i(C) = \{C_1, \ldots, C_s, C_{s+1}, \ldots, C_{s'}\}$ and $S_i(D) = \{D_1, \ldots, D_t, D_{t+1}, \ldots, D_{t'}\}$ with $C_1\sigma, \ldots, C_s\sigma$ containing L and $D_1\sigma, \ldots, D_t\sigma$ containing $\neg L$.

$$\begin{aligned} P_i \triangleq\ & \{\mathsf{conclusion}(C_i\sigma, D_j\sigma) \mid 1 \leq i \leq s, 1 \leq j \leq t\} \\ & \cup \{C_{s+1}\sigma, \ldots, C_{s'}\sigma, D_{t+1}\sigma, \ldots, D_{t'}\sigma\} \end{aligned}$$

– If G_i is the conclusion of a factoring step from the clause C with a substitution σ, $P_i \triangleq S_i(C)\sigma$.

We further define S^- to be the result of removing all $\bot$s in the set of clauses S.

Claim (Aggregation Property). For each $1 \leq i \leq l$ that:

$(A1)$ $G_i = \bigvee P_i^-$;
$(A2)$ $X = \bigvee (S_i(X))^-$ for a premise X.

Proof. We proof the case by induction on the index i.

– Base case: $i = 1$.
In this case, all premises come from $\mathcal{N}$, and hence $S_1(C) = \Xi(C)$ for any premise C. So $(A2)$ holds due to the definition of $\Xi(\cdot)$. We consider the following cases for $(A1)$.
  - If G_1 is the conclusion of a resolution step from two clauses C, D from $\mathcal{N}$, $S_1(C) = \Xi(C)$ and $S_1(D) = \Xi(D)$. Let L' be a literal from C or D.
    - ∗ If $L' = L$ or $L' = \neg L$, L' doesn't appear in G_1 because G_1 is the conclusion by a resolution step on L'. According to the definition of P_1, it doesn't appear in P_1 or P_1^- either.
    - ∗ Otherwise, L' remains in G_1 and P_1^-.

    In addition, $\bot$ doesn't appear in G_1 or P_1^-. So G_1 and P_1^- contains exactly the same set of literals.
  - If G_1 is the conclusion of a factoring step from a clause C in $\mathcal{N}$, $\bigvee P_1^- = \bigvee (S_1(C))^-\sigma = C\sigma = G_1$.

– Inductive case: for each $j < i$, $G_i = \bigvee P_i^-$ and $X = \bigvee (S_i(X))^-$.
Property $(A2)$ holds because of the definition of $\Xi(\cdot)$ and the induction hypothesis on $(A1)$. For $(A1)$ we consider the following cases.
  - If G_i is the conclusion of a resolution step from two clauses C, D with the substitution σ on the literal L in C and the literal $\neg L$ in D. Let L' be a literal from $C\sigma$ or $D\sigma$. According to $(A2)$, L' appears in some clauses in $S_i(C)\sigma$ or $S_i(D)\sigma$.
    - ∗ $L' = L$ or $L' = \neg L$, L' doesn't appear in G_i because G_i is the conclusion by a resolution step on L'. According to the definition of P_i and P_i^-, it doesn't appear in P_i or P_i^- either.
    - ∗ Otherwise, L' remains in G_i and P_i^-.

    In addition, $\bot$ doesn't appear in G_i or P_i^-. So G_i and P_i^- contains exactly the same set of literals, and thus $G_i = \bigvee P_i^-$.

- If G_i is the conclusion of a factoring step from C with the substitution σ, we have that $G_i = C\sigma$ and $P_i = S_i(C)\sigma$. Then we have $G_i = \bigvee P_i^-$ according to $(A2)$.

Based on the aggregation property $(A1)$ and the fact that G_l is the empty clause, we have P_l contains only $\bot$.

Claim (Successor Property). For each $1 \leq i \leq l$ that *every clause in* P_i *has at least one successor or appears in* P_l.

Proof. We proceed by induction on the index i.

- Base case: $i = l$. The property holds trivially.
- Inductive case: for each $j > i$, every clause in P_j has at least one successor or appears in P_l.
  If $i < l$, G_i has a successor G_k whose existence is guaranteed by Definition 6 for some $k > i$.
  - G_k is the conclusion of a resolution step from G_i and C with the substitution σ on the literal L in G_i and the literal $\neg L$ in C. Let P_i be $\{D_1, \dots, D_t, D_{t+1}, \dots, D_{t'}\}$ with each clause in $\{D_1\sigma, \dots, D_t\sigma\}$ contains the literal L and each in $\{D_{t+1}, \dots, D_{t'}\}$ doesn't. According to the side aggregation property, we have $C\sigma = \bigvee (S_i(C))^-\sigma$. Since $C\sigma$ contains $\neg L$, so does $(S_i(C))^-$. Therefore, every clause in $\{D_1, \dots, D_t\}$ is able to be resolved with a clause in $S_i(C)$. So each of them is a premise of a clause in P_k and hence they have a successor. Since $D_{t+1}, \dots, D_{t'}$ are copied to P_k, then the property holds for them due to the induction hypothesis.
  - If G_k is the conclusion of a factoring step from G_i with the substitution σ, $P_k = P_i\sigma$. So every clause in P_i has a success in P_k. So the property also holds in this case.

Let $C \in \mathsf{sup}(G_1, \dots, G_l)$, then it is a premise of G_k for some $1 \leq k \leq l$. One can show as in the proof of the successor property that every clause C' in $\Xi(C)$ is either copied into P_k or has a conclusion in P_k. The successor property guarantees that we can extract a *weakened resolution derivation* $N_1, \dots, N_l$ s.t. $N_i \in P_i$ for each $1 \leq i \leq l$ and C' is a premise of N_k. A weakened derivation means each N_i can be either a conclusion as in Definition 6 or be copied from N_j for $j < i$. Every weakened derivation can be trivially rewritten into a resolution derivation by eliminating repeating clauses. Since $N_l = \bot$, $N_1, \dots, N_l$ is in fact a weakened proof of $\bot$. So C' is in the support of a proof of $\bot$ in $\Xi(\mathcal{N})$. We have hence proved $(C1)$.

Claim $(C2)$ can be proved analogously. Let $G_1, \dots, G_l$ be a resolution derivation of $Q(\mathbf{a})$ in $\mathcal{N}$. We can construct $P_1, \dots, P_l$ in the same way as described above. But in this case, the aggregation property implies that P_l is either $Q(\mathbf{a})$ or $\bot$. Let $C \in \mathsf{sup}(G_1, \dots, G_l)$, then it is a premise of G_k from $\mathcal{N}$ for some $1 \leq k \leq l$. For each $C' \in \Xi(C)$, there is a weakened resolution derivation $N_1, \dots, N_l$ s.t. $N_i \in P_i$ for each $1 \leq i \leq l$ and C' is a premise of N_k. So C' is in the support of a refutation or a derivation of $Q(\mathbf{a})$ in $\Xi(\mathcal{N})$.

B Datalog-based Approach

Theorem 2. *Let $\mathcal{K}$ be a datalog knowledge base and let F be a set of facts in $Mat(\mathcal{K})$. Then, a fact $P(\mathbf{a})$ (resp. a rule r) in $\mathcal{K}$ participates in an SLD proof of some fact in F iff $\bar{P}(\mathbf{a})$ (resp. $\mathbf{S}(\mathbf{c_r})$) is in $Mat(\Delta(\mathcal{K}, F))$.*

By slightly abuse of notations, for an atom $A = P(\mathbf{t})$, we define $\bar{A} = \bar{P}(\mathbf{t})$ in the following proof.

Proof (Proof of Theorem 2). We first prove the "only if" direction. Assume that a fact $P(\mathbf{a})$ (a rule r) participates in the following SLD proof of a fact $Q(\mathbf{c}) \in F$.

$$G_0 = \neg Q(\mathbf{c}) \overset{C_1,\theta_1}{\rightsquigarrow} G_1 \rightsquigarrow \ldots \overset{C_l,\theta_l}{\rightsquigarrow} G_l$$

Let $\theta = \theta_1 \ldots \theta_l$. W.o.l.g. we can assume that $\theta_i\theta = \theta$ for each $1 \leq i \leq l$. This can be obtained by renaming variables.

Claim (Entailment Property). For each $1 \leq i \leq l$, $\mathcal{K} \models (A_1 \wedge \ldots \wedge A_n)\theta$ where $G_i = \neg A_1 \vee \ldots \vee \neg A_n$.

Proof. We prove the property by induction on the index i.

- Base case: If $i = l$, it holds trivially.
- Inductive case: Assume the following conditions.

$$G_{i-1} = \neg A_1 \vee \ldots \vee \neg A_n \tag{11}$$
$$C_i = \neg B_1(\mathbf{x}_1) \vee \ldots \vee \neg B_m(\mathbf{x}_m) \vee H(\mathbf{x}) \tag{12}$$
$$\theta_i = \mathsf{MGU}(A_k, H(\mathbf{x})) \tag{13}$$

  Then G_i is the following clause

$$(\neg A_1 \vee \ldots \vee \neg A_{k-1} \vee \neg B_1(\mathbf{x}_1) \vee \ldots \neg B_m(\mathbf{x}_m) \vee \neg A_{k+1} \vee \neg A_n)\theta_i \tag{14}$$

  The induction hypothesis of i ensures that $\mathcal{K} \models G_i\theta$, we then prove that the property holds for $i - 1$ as well.
  Note that $\theta_i\theta = \theta$, then $\mathcal{K} \models (A_1 \wedge \ldots A_{k-1} \wedge A_{k+1} \wedge \ldots \wedge A_n)\theta$ and $\mathcal{K} \models (B_1(\mathbf{x}_1) \wedge \ldots \wedge B_m(\mathbf{x}_m))\theta$. Since $\mathcal{K} \models C_i$, then $\mathcal{K} \models H(\mathbf{x})\theta$. Because $A_k\theta_i = H(\mathbf{x})\theta_i$, then $\mathcal{K} \models (A_1 \wedge \ldots \wedge A_n)\theta$. So the property holds for $i - 1$ as well.

Claim (Derivable Property). For each $1 \leq i \leq l$, $\Delta(\mathcal{K}, F) \models (\bar{A}_1 \wedge \ldots \wedge \bar{A}_n)\theta$ where $G_i = \neg A_1 \vee \ldots \vee \neg A_n$.

Proof. We prove the property by induction on the index i.

- Base case: If $i = 0$, $\bar{Q}(\mathbf{c}) \in \Delta(\mathcal{K}, F)$ and then $\Delta(\mathcal{K}, F) \models \bar{Q}(\mathbf{c})$. So the property holds.

– Inductive case: Assume we have (11), (12) and (13) hold, then G_{i+1} is the clause shown in (14).
  The induction hypothesis of $i-1$ ensures that $\Delta(\mathcal{K}, F) \models (\bar{A}_1 \wedge \ldots \wedge \bar{A}_n)\theta$, we then prove the property also holds i.
  It is trivial that $(\neg A_1 \vee \ldots \vee \neg A_{k-1} \vee \neg B_1(\mathbf{x}_1) \vee \ldots \neg B_m(\mathbf{y}_m) \vee \neg A_{k+1} \vee \neg A_n)\theta$ is an instance of G_i, because $\theta_i\theta = \theta$. According to the induction hypothesis, we have $\Delta(\mathcal{K}, F) \models (\bar{A}_1 \wedge \ldots \wedge \bar{A}_n)\theta$. So it suffices to prove that $\Delta(\mathcal{K}, F) \models (\bar{B}_1(\mathbf{x}_1) \wedge \ldots \wedge \bar{B}_m(\mathbf{x}_m))\theta$.
  The entailment property implies that $\mathcal{K} \models (B_1(\mathbf{x}_1) \wedge \ldots \wedge B_m(\mathbf{x}_m))\theta$. So does $\Delta(\mathcal{K}, F)$. In addition, $\Delta(\mathcal{K}, F) \models \bar{A}_k\theta$ (i.e. $\Delta(\mathcal{K}, F) \models \bar{H}(\mathbf{x})\theta$). Together with the rules $\bar{H}(\mathbf{x}) \wedge B_1(\mathbf{x}_1) \wedge \ldots \wedge B_m(\mathbf{x}_m) \rightarrow \bar{B}_j(\mathbf{x}_j)$ for each $1 \leq j \leq m$ in $\Delta(\mathcal{K}, F)$, we have $\Delta(\mathcal{K}, F) \models (\bar{B}_1(\mathbf{x}_1) \wedge \ldots \wedge \bar{B}_m(\mathbf{x}_m))\theta$.

If $P(\mathbf{a})$ participates in the above SLD proof, there is a $1 \leq i \leq n$, s.t. $C_i = P(\mathbf{a})$. Then G_{i-1} contains $\neg E$ with $E\theta_i = P(\mathbf{a})$. So the derivable property entails that $\Delta(\mathcal{K}, F) \models \bar{P}(\mathbf{a})$, and hence $\bar{P}(\mathbf{a}) \in Mat(\Delta(\mathcal{K}, F))$.

If a rule r participates in the above SLD proof. Assume that C_i is the corresponding clause of r of the form (12) and G_{i-1} is in the form of (11). Then G_i is the clause presented in (14). Because $\mathcal{K} \models (B_1(\mathbf{x}_1) \wedge \ldots \wedge B_m(\mathbf{x}_m))\theta$, so does $\Delta(\mathcal{K}, F)$. In addition, the derivable property guarantees that $\Delta(\mathcal{K}, F) \models \bar{A}_k\theta$ (i.e. $\Delta(\mathcal{K}, F) \models \bar{H}(\mathbf{x})\theta$). Together with the rule $\bar{H}(\mathbf{x}) \wedge B_1(\mathbf{x}_1) \wedge \ldots \wedge B_m(\mathbf{x}_m) \rightarrow \mathbf{S}(\mathbf{c_r})$ in $\Delta(\mathcal{K}, F)$, we have $\Delta(\mathcal{K}, F) \models \mathbf{S}(\mathbf{c_r})$, and hence $\mathbf{S}(\mathbf{c_r}) \in Mat(\Delta(\mathcal{K}, F))$.

The other direction can be shown by that if a fact $\bar{P}(\mathbf{a})$ (a rule $\mathbf{S}(\mathbf{c_r})$) is in $Mat(\Delta(\mathcal{K}, F))$, then there exists a hyper-resolution of a fact $Q(\mathbf{c})$ in F that involves $P(\mathbf{a})$ (or r). Every such proof corresponds to a resolution proof using the same rules and facts, and can hence be transformed using Lemma 4 into an SLD proof of $q(\mathbf{c})$.

Finally, it is clear that $Mat(\Delta(\mathcal{K}, F))$ can be computed in polynomial time because datalog evaluation is polynomial to the size of data, and the arity of predicates in the knowledge base $\mathcal{K}$ is bounded.