Practical higher-order query answering over $Hi(\textit{DL-Lite}_{\mathcal{R}})$ knowledge bases

Maurizio Lenzerini, Lorenzo Lepore, Antonella Poggi

Dipartimento di Ingegneria Informatica, Automatica e
Gestionale "Antonio Ruberti"- Sapienza Università di Roma
Via Ariosto 25, I-00183 Roma, Italy
lastname@dis.uniroma1.it

Abstract. The language $Hi(\textit{DL-Lite}_{\mathcal{R}})$ is obtained from *DL-Lite*$_{\mathcal{R}}$ by adding meta-modeling features, and is equipped with a query language that is able to express higher-order queries. We investigate the problem of answering a particular class of such queries, called *instance higher-order queries* posed over $Hi(\textit{DL-Lite}_{\mathcal{R}})$ knowledge bases (KBs). The only existing algorithm for this problem is based on the idea of reducing the evaluation of a higher-order query Q over a $Hi(\textit{DL-Lite}_{\mathcal{R}})$ KB to the evaluation of a union of first-order queries over a *DL-Lite*$_{\mathcal{R}}$ KB, built from Q by instantiating all metavariables in all possible ways. Even though of polynomial time complexity with respect to the size of the KB, this algorithm turns out to be inefficient in practice. In this paper we present a new algorithm, called *Smart Binding Planner (SBP)*, that compiles Q into a program, that issues a sequence of first-order conjunctive queries, where each query has the goal of providing the bindings for meta-variables of the next ones, and the last one completes the process by computing the answers to Q. We also illustrate some experiments showing that, in practice, SBP is significantly more efficient than the previous approach.

1 Introduction

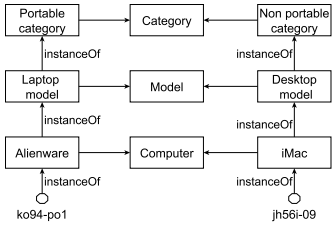
Description Logics (DLs) are popular formalisms for expressing ontologies, where an ontology is regarded as a formal specification of the concepts that are relevant in the domain of interest, together with the relationships between concepts. In many applications, the need arises of modeling and reasoning about metaconcepts and metaproperties. Roughly speaking, a metaconcept is a concept whose instances can be themselves concepts, and a metaproperty is a relationship between metaconcepts. Metaconcept representation is needed, for example, in formal ontology, where specific metaproperties (e.g., rigidity) are used to express relevant aspects of the intended meaning of the elements in an ontology. More generally, the idea of representing concepts and properties at the metalevel has been exploited in several sub-fields of Knowledge Representation and Computer Science, including semantic networks, early Frame-based and Description-based systems [10, 1], and conceptual modeling languages [12]. The notion of metaclass is also present in virtually all object-oriented languages, including modern programming languages (see, for instance, the Java class "class").

To see an example of metamodeling, consider the domain of computers, where we want to describe the properties of single computer machines (concept Computer in Figure 1), like the serial number and the manufacture date, the properties of the computer

models (Model), like the name and the country where computer machines of that models are produced, and the category (Category) of the various models. Thus, for instance, one could assert that a specific computer machine (e.g. ko94-pol) is an instance of Alienware (subconcept of Computer), where Alienware is an instance of Laptop model (subconcept of Model), and Laptop model is an instance of Portable category (subconcept of Category).

$Inst_C$(ko94-po1, Alienware)	$Inst_C$(jh56i-09, iMac)
$Inst_C$(Alienware, Laptop Model)	$Inst_C$(iMac, Desktop Model)
$Inst_C$(Laptop Model, Portable Category)	$Inst_C$(Desktop Model, Non Portable Category)
$Inst_R$(ko94-po1, Alienware, has_model)	$Inst_R$(jh56i-09, iMac, has_model)
Isa_C(Alienware, Computer)	Isa_C(iMac, Computer)
Isa_C(Computer, $Exists$(manufacture_date))	Isa_C(Computer, $Exists$(serial_number))
Isa_C($Exists$(serial_number),Computer)	Isa_C($Exists$(Inv(has_model)), Model)
Isa_C(Computer, $Exists$(has_model))	Isa_C($Exists$(has_model),Computer)
Isa_C(Laptop Model, Model)	Isa_C(Desktop Model, Model)
Isa_C(Model, $Exists$(has_category))	Isa_C($Exists$(has_category),Model)
Isa_C($Exists$(Inv(has_category)), Category)	Isa_C(Model, $Exists$(name))
Isa_C($Exists$(Inv(produced_in)), Country)	Isa_C(Model, $Exists$(produced_in))
Isa_C(Portable Category, Category)	Isa_C(Non Portable Category, Category)
$Disj_C$(Alienware, iMac)	$Disj_C$(Laptop Model, Desktop Model)
$Disj_C$(Portable Category, Non Portable Category)	

(a) List of assertions



(b) Diagrammatic representation

Fig. 1: $Hi(\textit{DL-Lite}_{\mathcal{R}})$ KB of computer domain

Obviously, the benefit of using metamodeling is greatly limited if one cannot express metaqueries on the knowledge base. A metaquery is an expression that combines predicates and metapredicates in order to extract complex patterns from the knowledge base. In particular, in a metaquery, variables can appear in predicate positions. In the example above, one might be interested in asking for all computer machines that are instances of a model that is an instance of Portable category, thus traversing three levels of the instance-of relation.

It is well-known that in logic, higher-order constructs are needed for a correct representation of concepts and properties at the meta-level. However, current research on Description Logics only rarely addresses the issue of extending the language with higher-order constructs (see, for instance, [2, 5, 11, 13, 9]). The starting point of our investigation is the work presented in [6, 7], where the authors propose a method to add medamodeling facilities to Description Logics, and study algorithms for answering metaqueries in a particular logic, called $Hi(\textit{DL-Lite}_{\mathcal{R}})$. The logic $Hi(\textit{DL-Lite}_{\mathcal{R}})$ is obtained from *DL-Lite*$_{\mathcal{R}}$ [4] by adding suitable constructs for metaconcepts and metaproperties modeling. Since the logics of the *DL-Lite* family are equipped with query answering algorithms with nice computational properties with respect to the size of the data, we believe that $Hi(\textit{DL-Lite}_{\mathcal{R}})$ is a good candidate for a language that can be used in ontology-based data access applications requiring to model metaconcepts and metaproperties.

It was shown in [6] that answering higher-order unions of conjunctive queries (HUCQs) over $Hi(\textit{DL-Lite}_{\mathcal{R}})$ KBs is in general intractable, while answering instance-based HUCQ (IHUCQ) has the same complexity as answering standard (i.e., fist-order) unions of conjunctive queries over *DL-Lite*$_{\mathcal{R}}$ KBs.

In particular, in [6] the authors present a query answering technique based on the idea of first transforming an IHUCQ Q over a $Hi(\textit{DL-Lite}_{\mathcal{R}})$ KBs $\mathcal{H}$ into a IHUCQ

Q' and then computing the certain answers to Q' over $\mathcal{H}$ seen as a standard *DL-Lite*$_\mathcal{R}$ KB. The goal of the technique was essentially to show that the data complexity class of query answering does not change if we add metamodeling to *DL-Lite*$_\mathcal{R}$. However, the technique presented in the paper turns out to be impractical in real world cases. Indeed, in most cases, the query answering algorithm computes the union of a huge number of metaground IHCQs, most of which can be in fact empty. This is due to the fact that the metagrounding is computed "blindly", i.e., by essentially instantiating in all possible ways all variables with the elements that are in predicate positions.

In this paper, we present a new algorithm for answering IHUCQs posed over Hi(*DL-Lite*$_\mathcal{R}$) knowledge bases, called *Smart Binding Planner (SBP)*. The basic idea of the algorithm is to compile the query q into a program expressed in a Datalog-like language, that issues a sequence of first-order conjunctive queries, where each query has the goal of providing the bindings for meta-predicates of the next ones, and the last one completes the process by computing the answers to q. We also illustrate some preliminary experiments showing that, in practice, SBP is significantly more efficient than the previous approach.

Note that our query language corresponds to a subset of SPARQL 1.1 interpreted under the OWL 2 direct semantics entailment regime [8], in particular the subset constituted by unions of conjunctive queries using only atoms on the instance-of relations. Thus, our algorithm can be seen as a first effective approach to answering such queries, but in the context where the KB has richer meta-modeling features than just punning.

The paper is organized as follows. In Section 2 we describe the Hi(*DL-Lite*$_\mathcal{R}$) logic. In section 3 we discuss the problem of expressing and evaluating IHUCQs in Hi(*DL-Lite*$_\mathcal{R}$), and we briefly illustrate the algorithm presented in [6]. In Section 4 we present our new algorithm, and Section 5 describes a set of experiments aiming at illustrating the behavior of our algorithm and at comparing it with the previous technique. Section 6 concludes the paper.

2 *Hi*(*DL-Lite*$_\mathcal{R}$)

In the following we recall the basics of Hi(*DL-Lite*$_\mathcal{R}$) [6], by first presenting its syntax and then its semantics. Following [6], we can characterize a traditional DL $\mathcal{L}$ by a set $OP(\mathcal{L})$ of *operators*, used to form concept and role expressions, and a set of $MP(\mathcal{L})$ of *metapredicates*, used to form assertions. Each operator and each metapredicate have an associated arity. If symbol S has arity n, then we write S/n to denote such a symbol and its arity. For *DL-Lite*$_\mathcal{R}$, we have: (*i*) OP(*DL-Lite*$_\mathcal{R}$) $= \{Inv/1, Exists/1\}$, which stand for "inverse role", and "existential restriction", (*ii*) MP(*DL-Lite*$_\mathcal{R}$) $= \{Inst_C/2, Inst_R/3, Isa_C/2, Isa_R/2, Disj_C/2, Disj_R/2\}$, which stand for instance, isa and disjoint assertions on both concepts and roles.

Syntax. Given a countably infinite alphabet $\mathcal{S}$ of *element names*, we inductively define the set of *expressions* for Hi(*DL-Lite*$_\mathcal{R}$), denoted by $\mathcal{E}_{\textit{DL-Lite}_\mathcal{R}}(\mathcal{S})$, over the alphabet $\mathcal{S}$ as follows:

- if $e \in \mathcal{S}$, then $e \in \mathcal{E}_{\textit{DL-Lite}_\mathcal{R}}(\mathcal{S})$;
- if $e \in \mathcal{E}_{\textit{DL-Lite}_\mathcal{R}}(\mathcal{S})$, and e is not of the form $Inv(e')$ (where e' is any expression), then $Inv(e) \in \mathcal{E}_{\textit{DL-Lite}_\mathcal{R}}(\mathcal{S})$;
- if $e \in \mathcal{E}_{\textit{DL-Lite}_\mathcal{R}}(\mathcal{S})$, then $Exists(e) \in \mathcal{E}_{\textit{DL-Lite}_\mathcal{R}}(\mathcal{S})$.

Intuitively, expressions denote elements, i.e., individuals, concepts and roles, of the knowledge base. The names in $\mathcal{S}$ are the symbols denoting the atomic elements, while the expressions denote either atomic elements, inverses of atomic elements (when interpreted as roles), or projections of atomic elements on either the first or the second component (again, when interpreted such elements as roles).

The basic building blocks of a $Hi(\textit{DL-Lite}_{\mathcal{R}})$ knowledge base are assertions. A *DL-Lite$_{\mathcal{R}}$-assertion*, or simply *assertion*, over the alphabet $\mathcal{S}$ for $Hi(\textit{DL-Lite}_{\mathcal{R}})$ is a statement of one of the forms

$$a_1(e_1, e_2),\ a_2(e_1, e_2, e_3)$$

where $a_1 \in MP(\textit{DL-Lite}_{\mathcal{R}})$ is either $Inst_C, Isa_C, Isa_R, Disj_C, Disj_R$, a_2 is $Inst_R$, and e_1, e_2, e_3 are expressions in $\mathcal{E}_{\textit{DL-Lite}_{\mathcal{R}}}(\mathcal{S})$. Thus, an assertion is simply an application of a metapredicate to a set of expressions, which intuitively means that an assertion is an axiom that predicates over a set of individuals, concepts or roles. A $Hi(\textit{DL-Lite}_{\mathcal{R}})$ *knowledge base (KB)* over $\mathcal{S}$ is a finite set of $Hi(\textit{DL-Lite}_{\mathcal{R}})$-assertions over $\mathcal{S}$.

For the sake of brevity, we do not include here the semantics of $Hi(\textit{DL-Lite}_{\mathcal{R}})$ and we refer the reader to [7]. We refer to [7] also for the notions of expressions occurring as respectively *object,concept* and *role argument* in one assertion.

3 Querying $Hi(\textit{DL-Lite}_{\mathcal{R}})$ knowledge bases

In this section we address the issue of querying $Hi(\textit{DL-Lite}_{\mathcal{R}})$ knowledge bases. Specifically, we define queries over $Hi(\textit{DL-Lite}_{\mathcal{R}})$ knowledge bases and report on the only query answering technique over such knowledge bases we are aware of. In the next section, we will then propose our new technique for query answering.

In order to define $Hi(\textit{DL-Lite}_{\mathcal{R}})$ queries, we first need to introduce the notion of "atom". We consider a countably infinite alphabet of variables $\mathcal{V}$, disjoint from $\mathcal{S}$, and a countably infinite alphabet of query predicates, each with its arity, disjoint from all other alphabets. An *atom* over $\mathcal{S}$ and $\mathcal{V}$ (or simply, an atom) has the form $a_1(e_1, e_2),\ a_2(e_1, e_2, e_3)$ where $a_1 \in MP(\textit{DL-Lite}_{\mathcal{R}})$ is either $Inst_C, Isa_C, Isa_R, Disj_C$ or $Disj_R$, a_2 is $Inst_R$, and each e_i is either an expression in $\mathcal{E}_{\textit{DL-Lite}_{\mathcal{R}}}(\mathcal{S})$ or a variable in $\mathcal{V}$, i.e., $e_i \in \mathcal{E}_{\textit{DL-Lite}_{\mathcal{R}}}(\mathcal{S}) \cup \mathcal{V}$. In the above assertions, a_1 and a_2 are called the predicates of the atom.

A *higher-order conjunctive query (HCQ)* of arity n is a formula of the form $q(u_1, \ldots, u_n) \leftarrow a_1, \ldots, a_m$ where $n \geq 0$, $m \geq 1$, q is a query predicate of arity n (which is also the arity of the query), each a_i is an atom, and each u_i is either in $\mathcal{V}$ and occurs in some a_j, or it is in $\mathcal{E}_{\textit{DL-Lite}_{\mathcal{R}}}(\mathcal{S})$. As usual, the tuple $\boldsymbol{u} = (u_1, \ldots, u_n)$ is called the *target of the query* q and the variables in $\boldsymbol{u}$ are its *distinguished variables*. The other variables of the query are called *existential variables*. Also, similarly to names within assertions, we say that a variable *occurs as concept argument* or *role argument within an atom*, depending on its position. Variables occurring as concept or role arguments are called *metavariables*. A *higher-order union of conjunctive queries (HUCQ)* of arity n is a set of HCQs of arity n with the same query predicate. Also a HUCQ is *metaground* if it does not contain any metavariable.

A HCQ is called *instance higher-order conjunctive query (IHCQ)* if each of its atoms has $Inst_C$ and $Inst_R$ as predicate. A HUCQ containing only IHCQs is called *instance higher-order union of conjunctive queries (IHUCQ)*.

Finally, a higher-order query is called *Boolean* if it has arity 0.

Example 1. Consider the $Hi(\textit{DL-Lite}_{\mathcal{R}})$ KB $\mathcal{H}$ in Figure 1. Suppose one needs to retrieve all items, together with their model and the model category, but only if the model category is classified as 'Portable Computer'. The IHCQ expressing this need is the following:

$$q(x, y, z) \leftarrow \mathit{Inst}_C(x, y), \mathit{Inst}_C(y, z), \mathit{Inst}_C(z, \text{'Portable Category'})$$

□

In order to define the semantics of queries, we now introduce the notion of assignment. Indeed, to interpret non-ground terms, we need assignments over interpretations, where an *assignment* μ over $\langle \Sigma, \mathcal{I}_o \rangle$ is a function $\mu : \mathcal{V} \rightarrow \Delta$. Given an interpretation $\mathcal{I} = \langle \Sigma, \mathcal{I}_o \rangle$ and an assignment μ over $\mathcal{I}$, the interpretation of terms is specified by the function $(\cdot)^{\mathcal{I}_o,\mu} : \mathcal{E}_{\textit{DL-Lite}_{\mathcal{R}}}(\mathcal{S}) \cup \mathcal{V} \rightarrow \Delta$ defined as follows: (*i*) if $e \in \mathcal{S}$ then $e^{\mathcal{I}_o,\mu} = e^{\mathcal{I}_o}$; (*ii*) if $e \in \mathcal{V}$ then $e^{\mathcal{I}_o,\mu} = \mu(e)$; (*iii*) $\mathit{op}(e)^{\mathcal{I}_o,\mu} = \mathit{op}^{\mathcal{I}_o}(e^{\mathcal{I}_o,\mu})$.

Finally, we define the semantics of atoms, by defining the notion of satisfaction of an atom with respect to an interpretation $\mathcal{I}$ and an assignment μ over $\mathcal{I}$ as follows:

- $\mathcal{I}, \mu \models \mathit{Inst}_C(e_1, e_2)$ if $e_1^{\mathcal{I}_o,\mu} \in (e_2^{\mathcal{I}_o,\mu})^{\mathcal{I}_c}$;
- $\mathcal{I}, \mu \models \mathit{Inst}_R(e_1, e_2, e_3)$ if $\langle e_1^{\mathcal{I}_o,\mu}, e_2^{\mathcal{I}_o,\mu} \rangle \in (e_3^{\mathcal{I}_o,\mu})^{\mathcal{I}_r}$;
- $\mathcal{I}, \mu \models \mathit{Isa}_C(e_1, e_2)$ if $(e_1^{\mathcal{I}_o,\mu})^{\mathcal{I}_c} \subseteq (e_2^{\mathcal{I}_o,\mu})^{\mathcal{I}_c}$;
- $\mathcal{I}, \mu \models \mathit{Isa}_R(e_1, e_2)$ if $(e_1^{\mathcal{I}_o,\mu})^{\mathcal{I}_r} \subseteq (e_2^{\mathcal{I}_o,\mu})^{\mathcal{I}_r}$;
- $\mathcal{I}, \mu \models \mathit{Disj}_C(e_1, e_2)$ if $(e_1^{\mathcal{I}_o,\mu})^{\mathcal{I}_c} \cap (e_2^{\mathcal{I}_o,\mu})^{\mathcal{I}_c} = \emptyset$;
- $\mathcal{I}, \mu \models \mathit{Disj}_R(e_1, e_2)$ if $(e_1^{\mathcal{I}_o,\mu})^{\mathcal{I}_r} \cap (e_2^{\mathcal{I}_o,\mu})^{\mathcal{I}_r} = \emptyset$.

Let $\mathcal{I}$ be an interpretation and μ an assignment over $\mathcal{I}$. A Boolean HCQ q of the form $q \leftarrow a_1, \ldots, a_n$ is *satisfied* in $\mathcal{I}, \mu$ if every query atom a_i is satisfied in $\mathcal{I}, \mu$. Given a Boolean HCQ q and a $Hi(\textit{DL-Lite}_{\mathcal{R}})$ KB $\mathcal{H}$, we say that q is *logically implied* by $\mathcal{H}$ (denoted by $\mathcal{H} \models q$) if for each model $\mathcal{I}$ of $\mathcal{H}$ there exists an assignment μ such that q is satisfied by $\mathcal{I}, \mu$. Given a non-Boolean HCQ q of the form $q(e_1, \ldots, e_n) \leftarrow a_1, \ldots, a_m$, a grounding substitution of q is a substitution θ such that $e_1\theta, \ldots, e_n\theta$ are ground terms. We call $e_1\theta, \ldots, e_n\theta$ a grounding tuple. The set of *certain answers* to q in $\mathcal{H}$, denoted by $\mathit{cert}(q, \mathcal{H})$, is the set of grounding tuples $e_1\theta, \ldots, e_n\theta$ that make the Boolean query $q_\theta \leftarrow a_1\theta, \ldots, a_n\theta$ logically implied by $\mathcal{H}$. These notions extend immediately to HUCQs.

As we said in the introduction, in [6] the authors present a query answering technique based on the idea of first transforming a IHUCQ Q over a $Hi(\textit{DL-Lite}_{\mathcal{R}})$ KBs $\mathcal{H}$ into a metaground IHUCQ Q' and then computing the certain answers to Q' over $\mathcal{H}$ seen as a standard *DL-Lite*$_{\mathcal{R}}$ KB.[1] It was shown that query answering with such a technique is in AC^0 w.r.t. the number of instance assertions, in PTIME w.r.t. KB complexity, and NP-complete w.r.t. combined complexity. While describing, in the following, the technique of [6] in more details, we introduce notions that we use in the next sections.

Given a $Hi(\textit{DL-Lite}_{\mathcal{R}})$ KB $\mathcal{H}$, we respectively denote by $\mathit{Roles}(\mathcal{H})$ and $\mathit{Concepts}(\mathcal{H})$ the sets $\mathit{Roles}(\mathcal{H}) = \{e, \mathit{Inv}(e) | e \in \mathcal{E}_{\textit{DL-Lite}_{\mathcal{R}}}(\mathcal{S})$ and e occurs as role

[1] In fact, in [6], the authors consider KBs equipped with mappings, and therefore their technique aims at rewriting the initial query over the actual data sources. Here, we ignore this aspect and adapt their algorithm to our scenario.

argument in $\mathcal{H}\}$ and $Concepts(\mathcal{H}) = \{e \mid e \in \mathcal{E}_{\textit{DL-Lite}_\mathcal{R}}(\mathcal{S})$ and e occurs as concept argument in $\mathcal{H}\} \cup \{Exists(e), Exists(Inv(e)) \mid e \in \mathcal{E}_{\textit{DL-Lite}_\mathcal{R}}(\mathcal{S})$ and e occurs as role argument in $\mathcal{H}\}$. Given two IHCQs q, q' and a KB $\mathcal{H}$, we say that q' is a *partial metagrounding of q with respect to* $\mathcal{H}$ if $q' = \sigma(q)$ where σ is a partial substitution of the metavariables of q such that for each metavariable x of q, either $\sigma(x) = x$, or: (*i*) if x occurs in concept position in q, then $\sigma(x) \in Concepts(\mathcal{H})$; (*ii*) if x occurs in role position in q, then $\sigma(x) \in Roles(\mathcal{H})$. Given a IHCQ q and a KB $\mathcal{H}$, we denote by $PMG(q, \mathcal{H})$ the IHUCQ constituted by the union of all the partial metagroundings of q w.r.t. $\mathcal{H}$. Moreover given a IHUCQ Q and a KB $\mathcal{H}$, we denote by $PMG(Q, \mathcal{H})$ the IHUCQ formed by $\bigcup_{q \in Q} PMG(q, \mathcal{H})$.

With these notions in place, the algorithm of [6], which we denote as COMPUTEPMG, can be described as follows. Given a IHUCQ Q and a $Hi(\textit{DL-Lite}_\mathcal{R})$ KB $\mathcal{H}$, it first computes the query $PMG(Q, \mathcal{H})$ and then returns the certain answers to the query $PMG(Q, \mathcal{H})$ with respect to $\mathcal{H}$. It is worth noting that, although polynomial w.r.t. the number of instance assertions in $\mathcal{H}$, COMPUTEPMG can be very inefficient in practice, in the sense that it computes the union of a huge number of IHCQs, most of which can be in fact empty. This is due to the fact that the partial metagrounding is computed "blindly", i.e., by instantiating in all possible ways, all metavariables with elements of $Concepts(\mathcal{H})$ and $Roles(\mathcal{H})$.

4 New algorithm for answering IHUCQs in $Hi(\textit{DL-Lite}_\mathcal{R})$

In this section we present the *Smart Binding Planner (*SBP*)* algorithm, aiming at reducing the number of metaground queries to be evaluated. The algorithm is based on a process that computes a sequence of metaground IHCQs, where each query has the goal of providing both the answers to the query, and the bindings for metavariables of the next ones.

Before delving into the details of the SBP algorithm, we shortly sketch its three main steps. First, it splits the query into a sequence of subqueries (function SPLITANDORDER) such that the evaluation of the i-th subquery provides the bindings to instantiate the metavariables of the $(i+1)$-th subquery. Second, based on such subqueries ordering, it builds a program (function DHQPSYNTHESIS), expressed in a specific language named *Datalog-based Higher Order Query Plan* (DHQP). Then, as third and final step, it evaluates the DHQP program (function EVALUATEDHQP).

In the following, we start by presenting the DHQP language and illustrating the function EVALUATEDHQP. We then define the two other functions that are used by SBP, namely the functions SPLITANDORDER and DHQPSYNTHESIS, and finally we present the complete algorithm SBP, and discuss its properties.

4.1 The DHQP language

Let $\mathcal{A}_\Gamma$ and $\mathcal{A}_I$ be two pairwise disjoint alphabets, disjoint from $\mathcal{S}$ and $\mathcal{V}$, called, respectively, the alphabet of *bridge predicates* and the alphabet of *intensional predicates*, each one with an associated arity. Intensional and bridge predicates are used to form a DHQP program, which, intuitively, is constitued by an ordered set of Datalog-like rules whose head predicate is an intensional predicate in $\mathcal{A}_I$, and whose body contains an

atom with a bridge predicate in $\mathcal{A}_\Gamma$. Every bridge predicate γ of arity n has an associated IHCQ of arity n, denoted $def(\gamma)$. Moreover, some of the n arguments of each bridge predicate γ are classified as "to-be-bound". When we write a bridge predicate, we indicate the variables corresponding to such arguments by underlining them. Intuitively, the underlined variables of the bridge predicate γ are those variables that are to be bound to ground expressions whenever we want to evaluate the query $def(\gamma)$. Also, the bridge predicate is used to denote the relation where we store the certain answers of a set of metaground IHCQs, each one obtained by $def(\gamma)$ by substituting the underlined variables of γ with a ground expression. Variables of the query $def(\gamma)$ that appear underlined in the corresponding bridge predicate γ are called *input variables* of the query. Note that, when we write the query predicate of $def(\gamma)$, we underline the query input variables.

We now provide the definition of the syntax of a DHQP program. Let m be a non-negative integer, let Γ be a sequence $(\gamma_0, \ldots, \gamma_m)$ of $m+1$ bridge predicates in $\mathcal{A}_\Gamma$, and let I be a sequence $(I_0, \ldots, I_m)$ of $m+1$ intensional predicates in $\mathcal{A}_I$.

A DHQP *program P over Γ and I* is a sequence of $m+1$ DHQP *rules* $r_0, r_1, \ldots, r_m$, such that

- r_0, called the *base rule* of P, has the form $I_0(\boldsymbol{v}) \leftarrow \gamma_0(\boldsymbol{w})$, where every variable in $\boldsymbol{v}$ occurs in $\boldsymbol{w}$.
- for each $1 \leq j \leq m$, rule r_j has the form $I_j(\boldsymbol{v}) \leftarrow I_{j-1}(\boldsymbol{u}), \gamma_j(\boldsymbol{w})$, where every variable in $\boldsymbol{v}$ occurs in $\boldsymbol{u}$ or $\boldsymbol{w}$, and every variable in $\boldsymbol{u}$ which is not an underlined variable in $\boldsymbol{w}$ appears in $\boldsymbol{v}$.

For every $0 \leq j \leq m$, we say that r_j defines I_j using γ_j. In the case of $1 \leq j \leq m$, we say that r_j defines I_j using γ_j based on I_{j-1}.

Example 2. Let $I = (I_0/1, I_1/2, I_2/3)$, and $\Gamma = (\gamma_0/1, \gamma_1/2, \gamma_2/2)$, with q_{γ_0}, q_{γ_1}, and q_{γ_2} defined as follows:

- $q_{\gamma_0}(z) = Inst_C(z, \text{Portable Category})$,
- $q_{\gamma_1}(\underline{z}, y) = Inst_C(y, z)$,
- $q_{\gamma_2}(\underline{y}, y) = Inst_C(x, y)$.

The following is a DHQP program over Γ and I, called P:

$$
\begin{aligned}
&r_0 : I_0(z) \leftarrow \gamma_0(z)\\
&r_1 : I_1(y, z) \leftarrow I_0(z), \gamma_1(\underline{z}, y)\\
&r_2 : I_2(x, y, z) \leftarrow I_1(y, z), \gamma_2(\underline{y}, x)
\end{aligned}
$$

□

We now turn our attention to the semantics of a DHQP program. Toward this goal, we introduce the notion of *instantiation of an IHCQ*. Given an IHCQ q, an n-tuple $\boldsymbol{x}$ of variables of q, and an n-tuple $\boldsymbol{t}$ of expressions in $\mathcal{E}_{\textit{DL-Lite}_\mathcal{R}}(\mathcal{S})$, the *$\boldsymbol{x}$-instantiation of q with $\boldsymbol{t}$*, denoted $INST(q, \boldsymbol{x} \leftarrow \boldsymbol{t})$, is the IHCQ obtained from q by substituting each x_i in $\boldsymbol{x}$ with the expression t_i in $\boldsymbol{t}$, for $i \in \{1, \ldots, n\}$. Note that a partial metagrounding of q with respect to $\mathcal{H}$ is an $\boldsymbol{x}$-instantiation of q with any n-tuple $\boldsymbol{t} = (t_1, t_2, \ldots, t_n)$, where $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ contains only metavariables in q, and such that, for every $i \in \{1, \ldots, n\}$, $t_i \in \mathit{Concept}(\mathcal{H})$ if x_i occurs as concept argument within q, and $t_i \in \mathit{Role}(\mathcal{H})$ if x_i occurs as role argument.

Let P be a DHQP program constituted by the rules $(r_0, \ldots, r_m)$. We specify the semantics of P operationally, i.e., we define an algorithm, called EVALUATEDHQP,

that, given as input a DHQP program P, and a $Hi(\textit{DL-Lite}_{\mathcal{R}})$ KB $\mathcal{H}$, computes the result of evaluating P with respect to a $\mathcal{H}$ as follows:

- Consider rule r_0, and compute the extension of I_0 as the certain answers to the query $PMG(q_{\gamma_0}, \mathcal{H})$ over the KB $\mathcal{H}$.
- For each $1 \leq i \leq m$, consider rule r_i, and compute the extension of I_i by evaluating the Datalog rule $I_i(\boldsymbol{v}) \leftarrow I_{i-1}(\boldsymbol{u}), \gamma_i(\boldsymbol{w})$, where the extension of I_{i-1} is the result computed when considering rule r_{i-1}, and the extension of γ_i is constituted by the certain answers of the query $PMG(q', \mathcal{H})$ over the KB $\mathcal{H}$, with q' constructed as follows: $\bigcup_{\boldsymbol{t} \in \pi_{\boldsymbol{w'}}(I_{i-1})} INST(q_{\gamma_i}, \boldsymbol{w'} \leftarrow \boldsymbol{t})$, where $\boldsymbol{w'}$ denotes the input variables of q_{γ_i} occurring both in $\boldsymbol{u}$ and in $\boldsymbol{w}$, and $\pi_{\boldsymbol{w'}}(R)$ denotes the projection of relation R onto the arguments corresponding to $\boldsymbol{w'}$.
- Return the extension of the predicate I_m as the result of evaluating P over $\mathcal{H}$.

Example 3. Consider the DHQP program P described in Example 2, and the KB $\mathcal{H}$ in Figure 1. The algorithm EVALUATEDHQP$(P, \mathcal{H})$ proceeds as follows. First, it considers rule r_0 and computes the certain answers of the metaground query q_{γ_0} with respect to $\mathcal{H}$, which, in this case, produces the following result: $\{(\text{Laptop Model})\}$. By means of the rule r_0, this result becomes also the extension of I_0. Then, it considers rule r_1, and, for each tuple of I_0 (in this case, only one) evaluates the query resulting from the z-instantiation of q_{γ_1} with the tuple itself; this means that the certain answers to the metaground query $q(y) \leftarrow \mathit{Inst}_C(y, \text{Laptop Model})$ (i.e., $\{(\text{Alienware})\}$) are computed and stored in the relation $\gamma_1(z, y)$. Such certain answers are used in rule r_1 to compute the extension of I_1, which becomes $\{(\text{Laptop Model}, \text{Alienware})\}$. Finally, the algorithm considers r_2, and computes the final result $\{(\text{ko94-po1}, \text{ Alienware}, \text{ Laptop Model})\}$. □

4.2 The function SPLITANDORDER

Given an IHCQ q with distinguished variables $\boldsymbol{x}$, SPLITANDORDER returns a sequence of subqueries of q by proceeding as follows.

1. It builds the *dependency graph* of q. The dependency graph of q is a directed graph whose nodes are the atoms of q, and the edges are formed according to the following rule: there is an edge from an atom a_1 to an atom a_2 if it at least one of the following is verified: (*i*) a_2 *depends on* a_1, i.e., if the expression occurring as predicate argument in a_2 is a variable occurring as object argument in a_1; (*ii*) there exists a variable x of Q that is not a metavariable and x occurs as object argument in both a_1 and a_2 (note that in this case, there will be an edge also from a_1 to a_2). Intuitively, a_2 depends on a_1, if by evaluating the query with body a_1, one obtains bindings for the metavariable of a_2, which allow to instantiate and then evaluate the subquery with body a_2.
2. It assigns to each atom a a *depth* $d(a)$ by using the following breadth-first strategy:
   (a) $k \leftarrow 0$;
   (b) do the following until the graph is empty:
      2.1 assign k to each atom n occurring in a strongly connected component C such that every atom in C has all of its incoming edges coming from C;
      2.2 remove from the graph all the considered atoms and their edges;
      2.3 $k \leftarrow k + 1$.

3. For every atom a, it marks every variable that is a metavariable and occurs in an atom a' such that $d(a') < d(a)$. Intuitively, if a variable within an atom is marked, then bindings for such a variable will be provided in order to evaluate the query with body that atom by using a standard *DL-Lite*$_{\mathcal{R}}$ reasoner.
4. For every $i \in \{0, m\}$, where m is the maximal atom depth, it defines an IHCQ q_{γ_i} whose arity is the number of variables occurring in some atom at depth i, and whose form is $q_{\gamma_i}(\underline{\boldsymbol{u}_{\boldsymbol{\gamma_i}}}, \boldsymbol{u_i}) \leftarrow b_{\gamma_i}$, where $\boldsymbol{u}_{\boldsymbol{\gamma_i}}$ contains all variables that are marked in some atom at depth i, $\boldsymbol{u_i}$ contains all unmarked variables occurring in some atom at depth i that also belong to $\boldsymbol{x}$ or, for $i < m$ are marked in some atom at depth $i + 1$, and b_{γ_i} is the conjunction of all atoms at depth i. Intuitively, atoms having the same depth will be evaluated in the same query.

The queries $q_{\gamma_0}, \ldots, q_{\gamma_m}$ built by SPLITANDORDER(q) will be used by the function DHQPSYNTHESIS to construct the DHQP program whose evaluation will return the answers to q.

Example 4. Consider the IHCQ presented in Example 1. The result of the SPLITANDORDER function is the sequence $B = (q_{\gamma_0}, q_{\gamma_1}, q_{\gamma_2})$ presented in Example 2. □

4.3 The function DHQPSYNTHESIS

Given a IHCQ q with target tuple $\boldsymbol{x}$, and the sequence $(q_{\gamma_0}, q_{\gamma_1}, \ldots, q_{\gamma_m})$ obtained by SPLITANDORDER(q), DHQPSYNTHESIS builds a DHQP program constituted by $m + 1$ as follows:

- The base rule has the form $I_0(\boldsymbol{v}) \leftarrow \gamma_0(\boldsymbol{w})$, where $def(\gamma_0) = q_{\gamma_0}$, each variable of $\boldsymbol{w}$ that is an input variable of q_{γ_0} is underlined, and $\boldsymbol{v}$ is such that: if $m = 0$, then $\boldsymbol{v}$ coincides with $\boldsymbol{x}$, otherwise $\boldsymbol{v}$ is constituted by all the variables in $\boldsymbol{w}$ that are in $\boldsymbol{x}$, and all the variables in $\boldsymbol{w}$ in position of input arguments in q_{γ_1}.
- For $j \in \{1, \ldots, m\}$, the j-th rule defining I_j based on I_{j-1} has the form $I_j(\boldsymbol{v}) \leftarrow I_{j-1}(\boldsymbol{u}), \gamma_j(\boldsymbol{w})$ where $def(\gamma_j) = q_{\gamma_j}$, each variable of $\boldsymbol{w}$ that is an input variable of q_{γ_j} is underlined, and $\boldsymbol{v}$ is such that if $j = m$, then $\boldsymbol{v}$ coincides with $\boldsymbol{x}$, otherwise $\boldsymbol{v}$ is constituted by all the variables in $\boldsymbol{w}$ and $\boldsymbol{u}$ that are in $\boldsymbol{x}$, and all the variables in $\boldsymbol{w}$ and $\boldsymbol{u}$ in position of input arguments in $q_{\gamma_{j+1}}$.

Example 5. Consider the IHCQ Q presented in Example 1 and the sequence of queries B obtained by executing SPLITANDORDER(Q), as shown in Example 4. It is easy to see that, by executing DHQPSYNTHESIS($\boldsymbol{x}$,B), where $\boldsymbol{x} = (x, y, z)$ is the target tuple of Q, we obtain the DHQP program presented in Example 2. □

4.4 The SBP Algorithm

We are now ready to describe the algorithm SBP. As shown in the following, the algorithm iterates over the various IHCQs in IHUCQ input query Q. At each iteration, the IHCQ q under exam is first split into subqueries, that are ordered on the basis of bindings that the evaluation of a subquery provides for variables of subsequent subqueries. On the basis of the ordering of subqueries, q is then compiled into a DHQP program, and then such a program is evaluated over $\mathcal{H}$. The result of such evaluation constitutes the resulting set of answers R.

Algorithm SBP(Q,$\mathcal{H}$)
input IHUCQ Q of arity n, Hi(*DL-Lite*$_{\mathcal{R}}$) KB $\mathcal{H}$
output Set of n-tuples of expressons in $\mathcal{E}_{\textit{DL-Lite}_{\mathcal{R}}}(\mathcal{S})$
begin
  $R = \emptyset$
  for each $q \in Q$
    $B =$ SPLITANDORDER(q);
    $P =$ DHQPSYNTHESIS(q,B);
    $R = R \cup$ EVALUATEDHQP(P, $\mathcal{H}$)
  return R
end

The following theorem shows termination and correctness of the algorithm.

Theorem 1. *Given a Hi(DL-Lite$_{\mathcal{R}}$) KB $\mathcal{H}$ and an IHUCQ q over $\mathcal{H}$, the* SBP$(q, \mathcal{H})$ *terminates and computes the certain answers of Q over $\mathcal{H}$.*

It is also easy to characterize the complexity of SBP. The following theorem provides such a charaterization.

Theorem 2. *Answering IHUCQs over Hi(DL-Lite$_{\mathcal{R}}$) KBs with* SBP *is* PTIME *w.r.t. both instance and KB complexity, and NP-complete w.r.t. combined complexity.*

5 Experiments

In this section we show the results of a preliminary set of tests that were carried out to compare the performances of SBP with that of the algorithm COMPUTEPMG presented at the end of Section 3. Such set of tests compares the evaluation time of the IHCQ q of Example 1 over four different extensions to the KB $\mathcal{H}$ in Figure 1. The extensions we have taken into account are the following.

- **Extension** A. The ontology is obtained by adding to the ontology $\mathcal{H}$ 30000 assertions of the form $Inst_C$(k_i,Alienware) and 30000 assertions of the form $Inst_C$(k_i,iMac) where $k_i \neq k_j$ for $1 \leq i, j \leq 60000$ and $i \neq j$, and each k_i is a new name not occurring in any of the assertions of ontology $\mathcal{H}$;
- **Extension** B. The ontology is obtained by adding to A the following set of assertions $\{Isa_C$(Tablet Model, Model), Isa_C(iPad, Computer), $Inst_C$(Tablet Model, Portable Category), $Inst_C$(iPad, Tablet Model)$\}$;
- **Extension** C. The ontology is obtained by adding to B 30000 assertions of the form $Inst_C$(k_i,iPad) where $k_i \neq k_j$ for $1 \leq i, j \leq 30000$ and $i \neq j$, and each k_i is a new name not occurring in any of the assertions of ontology B;
- **Extension** D. The ontology is obtained by adding to A the following set of assertions $\{Isa_C$(Server Model, Model), Isa_C(RackR230, Computer), $Inst_C$(Server Model, Non Portable Category), $Inst_C$(RackR230, Server Model)$\}$.

The results of the tests are summarized in Table 1. Each row in the table shows the behaviors of the algorithms SBP and COMPUTEPMG when computing the certain answers to q over a given ontology. Columns in the table show: (*i*) the considered ontology, (*ii*) the number of concept expressions in its intensional assertions, (*iii*) the behaviour of SBP in terms of number of metaground IHCQs executed, and execution time, (*iv*) the beaviour of COMPUTEPMG).

		SBP		COMPUTEPMG	
$\mathcal{H}$	$\lvert Concepts(\mathcal{H})\rvert$	*# metaground IHCQs*	*time*[2]	*# metaground IHCQs*	*time*[2]
A	19	3	0, 16 sec.	361	14, 63sec.
B	21	5	0, 25 sec.	441	19, 27 sec.
C	21	5	0, 28 sec.	441	20, 64 sec.
D	21	3	0, 16 sec.	441	19, 03 sec.

Table 1: Comparison between SBP and COMPUTEPMG

First we notice that, when executed on the same ontology, SBP is up to 110 times faster then COMPUTEPMG. The reason of that improvement is due to the smart grounding instantiation of the metavariables of q performed by SBP. For instance, to evaluate q over the ontology A, COMPUTEPMG issues 361 metaground IHCQs obtained by considering all the substitutions in $Concepts(A)$ for the two metavariables occurring in the atoms of q, thus leading to $|Concepts(A)|^2 = 19^2 = 361$ metaground IHCQs that have to be evaluated whereas SBP induces the evaluation of only 3 metaground IHCQs. Second, we observe that the increase of the size of the ontology has different impacts on the two algorithms performances. Specifically, while the performance of COMPUTEPMG worsen as soon as assertions with new expressions are added to the intensional part of the ontology, this does not necessarily happen for SBP.

6 Conclusion

We have presented a new technique to answer IHCUQs over $Hi(\textit{DL-Lite}_{\mathcal{R}})$ KBs, which improves the one presented in [6] by avoiding the blind grounding of metavariables that may cause query answering to become impractical.

We have illustrated a first set of experiments showing the advantage of using our strategy. In addition to the set of simple tests described above, at the time of this writing we are running a set of tests involving real world ontologies. These new experiments confirm that answering IHUCQs with the COMPUTEPMG strategy can be very inefficient, and that SBP is a promising direction to pursue.

However, we are aware that there are other possible optimization that we can study to improve the work presented here. One notable example is the management of cycles in the dependency graph. Presently, we assign the same depth to all of the atoms occurring in a cycle but there are cases where breaking the cycle, thus assigning different depth to the involved atoms, can allow to further reduce the number of metaground queries that have to be considered during the evaluation of the correspondent program. Our future works will be focused on identifying those cases, and studying other possible optimization strategies.

Acknowledgements: Work partially supported by the EU under FP7, project Optique (Scalable End-user Access to Big Data), grant n. FP7-318338.

[2] The evaluation reported here refers to experiments carried out using MASTRO [3] as reasoner computing the certain answers to metaground queries posed to *DL-Lite*$_{\mathcal{R}}$ KBs

References

1. G. Attardi and M. Simi. Consistency and completeness of OMEGA, a logic for knowledge representation. In *Proc. of IJCAI'81*, pages 504–510, 1981.
2. L. Badea. Reifying concepts in description logics. In *Proc. of IJCAI'97*, pages 142–147, 1997.
3. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. F. Savo. The Mastro system for ontology-based data access. *Semantic Web J.*, 2(1):43–53, 2011.
4. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. of Automated Reasoning*, 39(3):385–429, 2007.
5. S. Colucci, T. Di Noia, E. Di Sciascio, F. M. Donini, and A. Ragone. Second-order description logics: Semantics, motivation, and a calculus. In *Proc. of DL 2010*, volume 573 of *CEUR,* `ceur-ws.org`, 2010.
6. G. De Giacomo, M. Lenzerini, and R. Rosati. Higher-order description logics for domain metamodeling. In *Proc. of AAAI 2011*, 2011.
7. F. Di Pinto, G. De Giacomo, M. Lenzerini, and R. Rosati. Ontology-based data access with dynamic TBoxes in *DL-Lite*. In *Proc. of AAAI 2012*, 2012.
8. B. Glimm and C. Ogbuji. Sparql 1.1 entailment regimes. W3C reccomendation, W3C, 2013. Available at `http://www.w3.org/TR/sparql11-entailment/`.
9. B. Glimm, S. Rudolph, and J. Volker. Integrated metamodeling and diagnosis in owl 2. In *Proc. of ISWC 2010*, pages 257–272, 2010.
10. F. Lehmann, editor. *Semantic Networks in Artificial Intelligence*. Pergamon Press, Oxford (United Kingdom), 1992.
11. B. Motik. On the properties of metamodeling in OWL. *J. of Logic and Computation*, 17(4):617–637, 2007.
12. J. Mylopoulos, P. A. Bernstein, and H. K. T. Wong. A language facility for designing database-intensive applications. *ACM Trans. on Database Systems*, 5(2):185–207, 1980.
13. J. Z. Pan and I. Horrocks. OWL FA: a metamodeling extension of OWL DL. In *Proc. of WWW 2006*, pages 1065–1066, 2006.