Query Answering in Object Oriented Knowledge Bases in Logic Programming: Description and Challenge for ASP

Vinay K. Chaudhri[1], Stijn Heymans[1], Michael Wessel[1], and Tran Cao Son[2]

[1] Artificial Intelligence Center, SRI International, Menlo Park, CA 94025, USA
[2] Computer Science Department, New Mexico State University, NM 88003, USA

Abstract. Research on developing efficient and scalable ASP solvers can substantially benefit by the availability of data sets to experiment with. `KB_Bio_101` contains knowledge from a biology textbook, has been developed as part of Project Halo, and has recently become available for research use. This is one of the largest KBs available in ASP and the reasoning with this KB is undecidable in general. We give a description of this data set and give ASP programs for a suite of queries that have been of practical interest. We explain why these queries pose significant practical challenges for the current ASP solvers.

1 Introduction

The `KB_Bio_101` represents knowledge from a textbook used for advanced high school and introductory college biology courses [15]. The KB was developed by SRI as part of their work for Project Halo[3] and contains a concept taxonomy for the whole textbook and detailed rules for 20 chapters of the textbook. SRI has tested the educational usefulness of this knowledge base in the context of an intelligent textbook called *Inquire* as it is used by students for learning material from one chapter[4].

The `KB_Bio_101` was originally developed using a knowledge representation and reasoning system called Knowledge Machine (KM) [7]. In recent work, we have added a conceptual modeling layer to Answer Set Programming (ASP), called *Object Oriented Knowledge Base* (OOKB), that is sufficient to capture the knowledge in `KB_Bio_101`. OOKB supports conceptual modeling primitives that are commonly found in description logic family of languages such as a facility to define classes and organize them into a hierarchy, define partitions, ability to define relations (also known as slots) and organize them into a relation hierarchy, support for domain, range and qualified number constraints, support for defining sufficient conditions of a class, and support for descriptive rules. The features in OOKB overlap with the features languages such as $\mathbb{FDNC}$ [8], *Datalog*$^{\pm}$ [4], and *ASP*^{fs} [1] in its support for function symbols. It differs in that the functions can be used to specify graph-structured objects which cannot be done in previous languages. The reasoning with OOKBs has been proven to be undecidable.

[3] `http://www.projecthalo.com/`
[4] `http://www.aaaivideos.org/2012/inquire_intelligent_textbook/`

In this paper, we propose to consider four queries of practical interest on `KB_Bio_101`. These queries have been found extremely useful in the context of *Inquire* and provide motivating drivers for the ASP solvers. This dataset presents an excellent opportunity for further development of ASP solvers for the following reasons.

- The recent developments in ASP suggest that ASP could potentially provide an ideal tool for large scale KBs. Yet, most of the KBs described in the literature are fairly small. `KB_Bio_101` provides a real-world ASP program that fits this bill.
- We note that `KB_Bio_101` contains rules with functions symbols and thus requires some tailoring or re-writing of rules of this type since the grounding is infinite. A simple example, that requires the encoding of object oriented knowledge bases with function symbols, is a KB consisting of a single class `person`, and a single relation `has-parent`, and a statement of the form "for each `person` there exists an instance of the `has-parent` relation between this person with another individual who is also a `person`".
- Several rules in an OOKB have variables and even though rules in `KB_Bio_101` follow a small number of axiom templates, the size of this knowlege base indicates that this could be a non-trivial task for new grounders.
- The `KB_Bio_101` cannot be expressed in commonly available decidable description logics because it contains graph structured descriptions. Efficient reasonign with graph structures is an area of active recent research [12, 13], and since there exists an export of `KB_Bio_101` for description logic systems also [?], this dataset provides an ideal usecase to explore the boundaries of decidable reasoning.
- The reasoning tasks of computing differences between two concepts and finding relationships between two individuals are computationally intensive tasks. The implementations of these tasks in *Inquire* rely on graph algorithms and trade completeness for efficiency. These tasks will present a tough challenges to ASP solvers.
- Last but not least, we believe that the KB could entice the development and/or experimentation with new solvers for extended classes of logic programs (e.g., language with existential quantifiers or function symbols).

In addition to the challenges listed above, it will be possible to define multiple new challenges of increasing difficulty that can be used to motivate further research and development of ASP solvers.

2 Background: Logic Programming and OOKB

2.1 Logic Programming

A logic program Π is a set of rules of the form

$$c \leftarrow a_1, \ldots, a_m, not\ a_{m+1}, \ldots, not\ a_n \tag{1}$$

where $0 \leq m \leq n$, each a_i is a literal of a first order language and $not\ a_j$, $m < j \leq n$, is called a negation as failure literal (or naf-literal). c can be a literal or omitted. A rule (program) is non-ground if it contains some variable; otherwise, it is a ground rule (program). When $n = 0$, the rule is called a *fact.* When c is omitted,

the rule is a *constraint*. Well-known notions such as substitution, the Herbrand universe $\mathcal{U}_\Pi$, and Herbrand base $\mathcal{B}_\Pi$ of a program Π are defined as usual.
The semantics of a program is defined over ground programs. For a ground rule r of the form (1), let $pos(r){=}\{a_1, \ldots, a_m\}$ and $neg(r){=}\{a_{m+1}, \ldots, a_n\}$. A set of ground literals X is consistent if there exists no atom a s.t. $\{a, \neg a\}{\subseteq}X$. A ground rule r is *satisfied* by X if (*i*) $neg(r){\cap}X{\neq}\emptyset$; (*ii*) $pos(r){\setminus}X{\neq}\emptyset$; or (*iii*) $c \in X$.
Let Π be a ground program. For a consistent set of ground literals S, the *reduct* of Π w.r.t. S, denoted by Π^S, is the program obtained from the set of all rules of Π by deleting (**i**) each rule that has a naf-literal *not* a in its body with $a \in S$, and (**ii**) all naf-literals in the bodies of the remaining rules. S is an *answer set* of Π [10] if it satisfies the following conditions: (**i**) If Π does not contain any naf-literal then S is the minimal set of ground literals satisfying all rules in Π; and (**ii**) If Π contains some naf-literal then S is an answer set of Π if S is the answer set of Π^S.
For a non-ground program Π, a set of literals in $\mathcal{B}_\Pi$ is an answer set of Π if it is an answer set of *ground*(Π) that is the set of all possible ground rules obtained from instantiating variables with terms in $\mathcal{U}_\Pi$. Π is *consistent* if it has an answer set. Π *entails* a ground literal a, $\Pi \models a$, if a belongs to every answer set of Π. For convenience in notation, we will make use of choice atoms as defined in [16] that can occur in a rule wherever a literal can. Answer sets of logic programs can be computed using answer set solvers (e.g., CLASP [9], **dlv** [6]).

2.2 Object-Oriented Knowledge Bases

We will now review the notion of an OOKB [5]. We note that an OOKB could be viewed as a logic program with function symbols and the language of OOKBs contains features that cannot be represented in previous investigated classes of function symbols such as $\mathbb{FDNC}$ [8], *Datalog*$^\pm$ [4], or ASP^{fs} [1]. In essense, an OOKBs is a logic programming consisting of the following components:

- *Taxonomic Knowledge:* This group of facts encodes the class hierarchy, the relation hierarchy, individual constants and their class membership. It contains ASP-atoms of the following form:

$$class(c) \quad (2)$$
$$individual(i) \quad (3)$$
$$subclass\_of(c_1, c_2) \quad (4)$$
$$disjoint(c_1, c_2) \quad (5)$$
$$instance\_of(i, c) \quad (6)$$
$$relation(r) \quad (7)$$
$$range(r, c) \quad (8)$$
$$domain(r, c) \quad (9)$$
$$subrelation\_of(r_1, r_2) \quad (10)$$
$$compose(r_1, r_2, r_3) \quad (11)$$
$$inverse(r_1, r_2) \quad (12)$$

  The predicate names are self-explanatory.
- *Descriptive statements:* Relationships between individuals are encoded in OOKB by descriptive statements of the form:

$$value(r, f(X), g(X)) \leftarrow instance\_of(X, c) \quad (13)$$
$$value(r, X, g(X)) \leftarrow instance\_of(X, c) \quad (14)$$

  where f and g are unary functions, called *Skolem functions*, such that $f \neq g$ and c is a class. (13) (or (14)) describes a relation value of individuals belonging to class c, encoded by the atom $value(r, f(X), g(X))$ (or $value(r, X, f(X))$. It states that for each individual X in c, $f(X)$ (or X) is related to $g(X)$ via the relation r. It is required that if f (or g) appears in (13) or (14), then the OOKB also contains the following rule

$$instance\_of(f(X), c_f) \leftarrow instance\_of(X, c) \quad \text{or} \tag{15}$$

$$instance\_of(g(X), c_g) \leftarrow instance\_of(X, c) \tag{16}$$

which specify the class of which $f(X)$ (resp. $g(X)$) is a member.

- *Cardinality constraints on relations:* OOKB allows cardinality constraints on relations to be specified by statements of the following form:

$$constraint(t, f(X), r, d, n) \leftarrow instance\_of(X, c) \tag{17}$$

where r is a relation, n is a non-negative integer, d and c are classes, and t can either be *min*, *max*, or *exact*. This constraint states that for each instance X of the class c, the set of values of relation r restricted on $f(X)$—which must occur in a relation value literal $value(r, f(X), g(X))$ of c—has minimal (resp. maximal, exactly) n values belonging to the class d. The head of (17) is called a constraint literal of c.

- *Sufficient conditions:* A *sufficient condition* of a class c defines sufficient conditions for membership of that class based on the relation values and constraints applicable to an instance:

$$instance\_of(X, c) \leftarrow Body(\boldsymbol{X}) \tag{18}$$

where $Body(\boldsymbol{X})$ is a conjunction of relation value literals, instance-of literals, constraint-literals of c, and X is a variable occurring in the body of the rule.

- *(In)Equality between individual terms:* The rules in this group specify in/equality between terms, which are constructable from Skolem functions and the variable X (t_1 and t_2), and have the followimg form:

$$eq(t_1, t_2) \leftarrow instance\_of(X, c) \tag{19}$$

$$neq(t_1, t_2) \leftarrow instance\_of(X, c) \tag{20}$$

- *Domain-independent axioms:* An OOKB also contains a set of domain-independent axioms Π_R for inheritance reasoning, reasoning about the relation values of individuals (rules (25)—(27)), in/equality between terms (rules (28)—(40)), and enforcing constraints (rules (42)—(47)).

$$subclass\_of(C, B) \leftarrow subclass\_of(C, A), subclass\_of(A, B). \tag{21}$$

$$instance\_of(X, C) \leftarrow instance\_of(X, D), subclass\_of(D, C). \tag{22}$$

$$disjoint(C, D) \leftarrow disjoint(D, C). \tag{23}$$

$$\neg instance\_of(X, C) \leftarrow instance\_of(X, D), disjoint(D, C). \tag{24}$$

$$value(U, X, Z) \leftarrow compose(S, T, U), value(S, X, Y), value(T, Y, Z). \tag{25}$$

$$value(T, X, Y) \leftarrow subrelation\_of(S, T), value(S, X, Y). \tag{26}$$

$$value(T, Y, X) \leftarrow inverse(S, T), value(S, X, Y). \tag{27}$$

$$eq(X, Y) \leftarrow eq(Y, X) \tag{28}$$

$$
\begin{aligned}
eq(X,Z) &\leftarrow eq(X,Y), eq(Y,Z), X \neq Z && (29)\\
&\leftarrow eq(X,Y), neq(X,Y) && (30)\\
\{substitute(X,Y)\} &\leftarrow eq(X,Y). && (31)\\
&\leftarrow eq(X,Y), \{substitute(X,Z) : eq(X,Z)\}0, && (32)\\
&\quad \{substitute(Y,Z) : eq(Y,Z)\}0. &&\\
&\leftarrow substitute(X,Y), substitute(X,Z), && (33)\\
&\quad X \neq Y, X \neq Z, Y \neq Z. && (34)\\
&\leftarrow substitute(X,Y), X \neq Y, neq(X,Y). && (35)\\
substitute(Y,Z) &\leftarrow substitute(X,Z), X \neq Z, eq(X,Y). && (36)\\
is\_substituted(X) &\leftarrow substitute(X,Y), X \neq Y. && (37)\\
substitute(X,X) &\leftarrow term(X), not\, is\_substituted(X). && (38)\\
term(X) &\leftarrow value(S,X,Y). && (39)\\
term(Y) &\leftarrow value(S,X,Y). && (40)\\
value_e(S,P,Q) &\leftarrow value(S,X,Y), substitute(X,P), substitute(Y,Q). && (41)\\
&\leftarrow value(S,X,Y), domain(S,C), not\, instance\_of(X,C). && (42)\\
&\leftarrow value(S,X,Y), range(S,C), not\, instance\_of(Y,C). && (43)\\
&\leftarrow constraint(min,Y,S,D,M), && (44)\\
&\quad \{value_e(S,Y,Z) : instance\_of(Z,D)\}\, M-1. &&\\
&\leftarrow constraint(max,Y,S,D,M), && (45)\\
&\quad M+1\{value_e(S,Y,Z) : instance\_of(Z,D)\}. &&\\
&\leftarrow constraint(exact,Y,S,D,M), && (46)\\
&\quad \{value_e(S,Y,Z) : instance\_of(Z,D)\}\, M-1. &&\\
&\leftarrow constraint(exact,Y,S,D,M), && (47)\\
&\quad M+1\{value_e(S,Y,Z) : instance\_of(Z,D)\}. &&
\end{aligned}
$$

An *OO-domain* is a collection of rules of the form (2)—(20). From now on, whenever we refer to an OOKB, we mean the prorgram $D \cup \Pi_R$, denoted by $KB(D)$, where D is the OO-domain of the OOKB[5].

2.3 `KB_Bio_101`: An OOKB Usage and Some Key Characteristics

The `KB_Bio_101` is available in OOKB format and access can be granted to it on request. The KB is based on an upper ontology called the Component Library [3]. The biologists used a knowledge authoring system called AURA to represent knowledge from a biology textbook. As an example, in Figure 1, we show an AURA graph corresponding to the example considered earlier in the paper. The white node labeled as **`Eukaryotic-Cell`** is the root node and represents the universally quantified variable X, whereas the other nodes shown in gray represent existentials, or the Skolem functions $f_n(X)$. The nodes labeled as **`has_part`** and **`is_inside`** represent the relation names. The authoring process in AURA can be abstractly characterized as involving three steps: *inherit, specialize and extend.*

[5] In [5], general OOKBs, that can contain arbitrary logic programming rules, were defined. The discussion in this paper is applicable to general OOKBs as well.

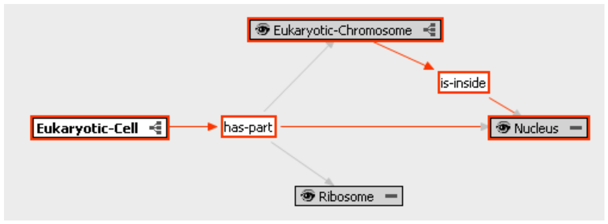

Fig. 1. Example graph for "Eukaryotic-Cell"

For example, the biologist creates the class `Eukaryotic-Cell` as a subclass of `Cell`. While doing so, the system would first inherit the relation values defined for `Cell` which in this case is a `Chromosome`, and show it in the graphical editor. The biologist then uses a gesture in the editor to specialize the inherited `Chromosome` to a `Eukaryotic-Chromosome`, and then introduces a new `Nucleus` and relates it to the `Eukaryotic-Chromosome`, via an `is-inside` relationship. The inherited `Chromosome` value for the `has-part` relationship, is thus, specialized to `Eukaryotic-Chromosome` and extended by connecting it to the `Nucleus` by using an `is-inside` relationship. The statistics about the size of the exported OOKB are summarized in Table 1. In total `KB_Bio_101` has more than 300,000 non-ground rules. It contains 746 individuals which are member of classes which represent constants of measurements, colors, shapes, quantity, priority, etc. The KB does not contain individuals of biology classes such as `cell`, `ribosome`, etc. For computing properties of an individual or comparing individuals, the input needs to specify the individuals.

classes	6430	*domain* constraints	449
individuals	746	*range* constraints	447
relations	455	*inverse* relation statements	442
subclass_of statements	6993	*compose* statements	431
subrelation_of statements	297	qualified number constraints	936
instance_of statements	714	sufficient conditions	198
disjoint-ness statements	18616	descriptive rules	6430
avg. number of Skolem functions in each descriptive rule	24	equality statements	108755

Table 1. Statistics on `KB_Bio_101`

3 Basic Queries in OOKBs

We will now describe the basic queries given an OOKB, say $KB(D)$. These basic queries play a central role in the educational application `Inquire` [14] which employs the knowledge encoded in `KB_Bio_101`. We divide these queries in four groups. The first type of queries focuses on the taxonomical hierarchy described by the KB. The second type of queries asks about the description of individuals of a class. The third type requests is about the fundamental distinctions and similarities between individuals from different classes. The fourth type of queries looks for specific ways that an individual in one class can affect an individual from another class. These types of queries are sufficiently expressive to represent frequently occurred questions in AP-biology exams. For example,

- is `eukaryotic cell` a subclass of `cell`?
- what is a `eukaryotic cell`?
- describe the differences and similarities between `mitochondrions` and `chloroplasts`;
- what process provides raw materials for the `citric acic cycle` during *cellular respiration*?
- in the absence of `oxygen`, `yeast cells` can obtain `energy` by *which process*?

Let Z be a set of literals of $KB(D)$, r be a relation, and i be an individual from a class c. $\mathcal{T}(i)$ denotes the set of terms constructable from Skolem functions and the individual i. We characterize the set of pairs in the relation r w.r.t. Z in $KB(D)$ by the set $V(r,i,c,Z) = \{(r,x,y) \mid value(r,x,y) \in Z, x,y \in \mathcal{T}(i)\}$ if $instance\_of(i,c) \in Z$; otherwise, $V(r,i,c,Z)=\emptyset$.

Definition 1 (Value set of an individual). *Let $KB(D)$ be an OOKB. For an answer set M of $KB(D)$, the* value set of an individual i at a class c *w.r.t. M, $\Sigma(i,c,M)$, is defined by $\Sigma(i,c,M) = \bigcup_{relation(r)\in M} V(i,c,r,M)$.*

Observe that the rules (29)—(41) indicate that $KB(D)$ can have multiple answer sets. Nevertheless, the structure of $KB(D)$ allows us to prove the following important property of answer sets of $KB(D)$.

Proposition 1. *Let $KB(D)$ be an OOKB. For every two answer sets M_1 and M_2 of $KB(D)$, every literal in $M_1 \setminus M_2$ has one of the following forms: (*i*) $substitute(x,y)$; (*ii*) $is\_substituted(x,y)$; or (*iii*) $value_e(r,x,y)$.*

The above proposition indicates that $\Sigma(i,c,M_1) = \Sigma(i,c,M_2)$ for arbitrary individual i and class c and answer sets M_1 and M_2 of $KB(D)$. The relationship between atoms of the form $value(r,x,y)$ and $value_e(r,x,y)$ is as follows.

Proposition 2. *Let $KB(D)$ be an OOKB, i an individual, and c a class. For every answer sets M of $KB(D)$, we have that $value_e(r,x,y) \in M$ iff there exists x', y' such that (*i*) M contains the following atoms $eq(x',x)$, $eq(y',y)$, $substitute(x',x)$, and $substitute(y',y)$; and (*ii*) $(r,x',y') \in \Sigma(i,c,M)$.*

The significance of these two propositions is that cautious reasoning about values of individuals at classes can be accomplished by computing *one* answer set of $KB(D)$. As we will see, the majority of queries is related to this type of reasoning. We next describe, for each query Q, an input program $I(Q)$ and a set $R(Q)$ of rules for computing the answer of Q. Throughout the section, KB denotes an arbitrary but fixed OOKB $KB(D)$ and $KB(Q) = KB(D) \cup I(Q) \cup R(Q)$.

3.1 Subsumption Between Classes ($\mathbf{Q_1}$)

Subsumption requires us to compute whether a class c_1 is subsumed by a class c_2, i.e., whether for each answer set M of $KB(Q_1)$, we have for each $instance\_of(x,c_1) \in M$ also $instance\_of(x,c_2) \in M$. We can answer this question by introducing introducing a fresh constant i in the OOKB and set $I(Q_1) = \{instance\_of(i,c_1)\}$. $R(Q_1)$ consists of a rule:

$$subclass\_of(c_1,c_2) \leftarrow instance\_of(i,c_2) \qquad (48)$$

Indeed, we then have that a class c_1 is subsumbed by c_2 iff for each answer set M of $KB(Q_1)$, $subclass\_of(c_1, c_2) \in M$. This conclusion comes from the following observations: (a) if $subclass\_of(c_1, c_2)$ is given as a fact (via (4)) then the subsumption is trivial; (b) if $subclass\_of(c_1, c_2)$ is not given as a fact then (48) is the only rule for deriving $subclass\_of(c_1, c_2)$ and for $subclass\_of(c_1, c_2) \in M$, it requires that $instance\_of(i, c_2) \in M$, i.e., each instance i of c_1 is also an instance of c_2. Proposition 1 can be extended to $KB(Q_1)$ and thus we only need to compute one answer set of $KB(Q_1)$. Note that this shows how, as in description logics, subsumption can be reduced to entailment in the OOKB framework. We can show that

Proposition 3. *If $KB(Q_1)$ has an answer set M and $subclass\_of(c_1, c_2) \in M$ then c_1 is subsumed by c_2.*

We note that computing answer sets of $KB(Q_1)$ is not a simple task (see [5]). In particular, the problem for `KB_Bio_101` is quite challenging due to its size and the potential infiniteness of the grounding program of $KB(Q_1)$.

3.2 Description of an Individual ($\mathbf{Q_2}$)

Queries about the description of an individual ask for a description of an individual of a class c, represented by a fresh constant i in the language of $KB(D)$. This query can be represented by the program $I(Q_2) = \{get\_value(i, c).instance\_of(i, c).\}$ where $get\_value(i, c)$ encodes the query of "inquiring about values of i at the class c." We will now discuss the answer to this query. Intuitively, a complete description of i should contain the following information:

- $C(c) = \{d \mid KB(D) \models subclass\_of(c, d)\}$, the classes from which i inherits its relation values; and
- its relation values, i.e., the triples in $\Sigma(i, c, M)$ where M is a given answer set of $R(Q_2)$.

Computing a complete description of i could be achieved by the following rules:

$$out\_member\_of(Y) \leftarrow get\_value(I, C), instance\_of(I, C), instance\_of(I, Y). \quad (49)$$

$$\begin{aligned} out\_value(R, X, Y) \leftarrow\ & get\_value(I, C), value(R, X, Y), relation(R), \\ & term\_of(X, I), term\_of(Y, I). \end{aligned} \quad (50)$$

where $term\_of(X, I)$ defines a term (X) that is constructable from Skolem functions and an individual (I), $out\_member\_of(d)$ indicates that i is an instance of the class d (i.e., $d \in C(c)$), and $out\_value(r, x, y)$ says that $KB(D) \models value(r, x, y)$. This answer is correct but may contain *too much* information for users of an OOKB who have knowledge about the class hierarchy. This is because the above description could also include values that i can inherit from the superclasses of c. This can be seen in the next example.

Example 1. Let us consider the class `Eukaryotic cell`. The description of this class contains 88 statements of the form (13)—(14) that involve 167 classes and 150 equality specifications. A first-level answer[6] computed using (49)–(50)

[6] Current solvers can only approximate the answer due to the infiniteness of the grounding program. We computed the answer by limiting the maximum nesting level for complex terms of the term to be 1 (e.g., the option `maxnesting` in **dlv**).

contains 9 atoms of the form $out\_member\_of(x)$ which indicate that a `eukaryotic cell` is also a `cell`, a `living entity`, a `physical object`, etc. In addition, there are 643 atoms of the form $out\_value(r, x, y)$ which contains inverse, composition, sub-relation, and the relation value defined in statements of the form (13)—(14) and those that are obtained by the rules (25)–(27).

The example highlights two challenges in computing the description of an individual. On the one hand, the perennial challenge lies in the infiniteness of the grounding program. On the other hand, for practical query answering applications, that use `KB_Bio_101`, deciding what to present to the user is another challenge. This topic is outside the scope of this paper and is one of our main current interests. It should be noted that because of the infiniteness of the ground KB, current ASP solvers can be used to approximate the answers. Whether this will result in acceptable performance, both in terms of the quality of the answers and the efficiency, remains a topic of research that we would like to study in the near future.

3.3 Comparing between Classes ($\mathbf{Q_3}$)

A comparison query takes the general form of "What are the differences/similarities between c_1 and c_2?" (e.g., "what are the differences between *chromosome* and *ribosome*?"). More specific versions of the query may ask for specific kinds of differences, e.g., structural differences.
The query can be represented and answered by (*i*) introducing two new constants i_1 and i_2 which are instances of c_1 and c_2, respectively; and (*ii*) identifying the differences and similarities presented in the descriptions of i_1 and i_2. We therefore encode $I(Q_3)$ using the following program:

$$instance\_of(i_1, c_1). \quad instance\_of(i_2, c_2). \quad comparison(i_1, c_1, i_2, c_2). \tag{51}$$

Let us first discuss the features that can be used in comparing individuals of two classes. Individuals from two classes can be distinguished from each other using different dimensions, either by their superclass relationship or by the relations defined for each class. More specifically, they can be differentiated from each other by the generalitation and/or specialitation between classes; or the properties of instances belonging to them. We will refer to these two dimensions as *class-dimension* and *instance-dimension*, respectively. We therefore define the following notions, given an answer set M of $KB(Q_3)$:

- *The set of similar classes between c_1 and c_2*: is the intersection between the set of superclasses of c_1 and of c_2

$$U(c_1, c_2) = C(c_1) \cap C(c_2) \tag{52}$$

- *The set of different classes between c_1 and c_2*: is the set difference between the set of superclasses of c_1 and of c_2

$$D(c_1, c_2) = (C(c_1) \setminus C(c_2)) \cup (C(c_2) \setminus C(c_1)) \tag{53}$$

where $C(c)$ denotes the set of superclasses of c.

We next discuss the question of what should be considered as a similar and/or different property between individuals of two different classes. Our formalization is motivated from the typical answers to this type of question such as an answer "a chromosome has a part as protein but a ribosome does not" to the query "what is the different between a chromosome and a ribosome?" This answer indicates that for each chromosome x there exists a part of x, say $f(x)$, which is a protein, i.e., $value(has\_part, x, f(x))$ and $instance\_of(f(x), protein)$ hold; furthermore, no part of a ribosome, say y, is a protein, i.e., there exists no g such that $value(has\_part, y, g(y))$ and $instance\_of(g(y), protein)$ hold.
For a set of literals M of $KB(Q_3)$ and a class c with $instance\_of(i, c) \in M$, let $T(i, c)$ be the set of triples (r, p, q) such that $(r, x, y) \in \Sigma(i, c, M)$, $instance\_of(x, p) \in M$, and $instance\_of(y, q) \in M$. p (q) is called the domain (range) of r if $(r, p, q) \in T(i, c)$. We define

- *The set of similar relations between* c_1 *and* c_2: is the set $R^s(c_1, c_2)$ of relations s such that (*i*) c_1 and c_2 are domain of s; (*ii*) c_1 and c_2 are range of s; or (*iii*) there exist (p, q) such that $(s, p, q) \in T(i_1, c_1) \cap T(i_2, c_2)$.
- *The set of different relations between* c_1 *and* c_2: is the set $R^d(c_1, c_2)$ of relations s such that (*i*) c_1 is and c_2 is not a domain of s or vice versa; (*ii*) c_1 is and c_2 is not a range of s vice versa; or (*iii*) there exist (p, q) such that $(s, p, q) \in (T(i_1, c_1) \setminus T(i_2, c_2)) \cup (T(i_2, c_2) \setminus T(i_1, c_1))$.

An answer to $\mathbf{Q_3}$ must contain information from $U(c_1, c_2)$, $D(c_1, c_2)$, $R^s(c_1, c_2)$, and $R^d(c_1, c_2)$. Computing $U(c_1, c_2)$ and $D(c_1, c_2)$ rely on the rules for determining the most specific classes among a group of classes which can easily be implemented using the naf-operator.
We now describe the set of rules $R(Q_3)$, dividing it into different groups. First, the set of rules for computing $U(c_1, c_2)$ is as follows:

$$\begin{aligned} shared(C, P, Q) \leftarrow\ & comparison(X, P, Y, Q), \\ & subclass\_of(P, C), subclass\_of(Q, C). \end{aligned} \tag{54}$$

The rule identifies the classes that are superclass of both c_1 and c_2. We can show that $KB(Q_3) \models shared(c, c_1, c_2)$ iff $c \in U(c_1, c_2)$.
The next set of rules is for computing $D(c_1, c_2)$.

$$\begin{aligned} dist(C, P, Q) \leftarrow\ & comparison(X, P, Y, Q), \\ & subclass\_of(P, C), not\ subclass\_of(Q, C). \end{aligned} \tag{55}$$

$$\begin{aligned} dist(C, P, Q) \leftarrow\ & comparison(X, P, Y, Q), \\ & not\ subclass\_of(P, C), subclass\_of(Q, C). \end{aligned} \tag{56}$$

The two rules identify the classes that are superclass of c_1 but not c_2 and vice versa. Again, we can show that $KB(Q_3) \models dist(c, c_1, c_2)$ iff $c \in D(c_1, c_2)$.
For computing $R^s(c_1, c_2)$ and $R^d(c_1, c_2)$, we need to compute the sets $T(i_1, c_1)$ and $T(i_2, c_2)$. For this purpose, we define two predicates t_1 and t_2 such that for every answer set M of $KB(Q_3)$, $t_k(s, p, q) \in M$ iff $(s, p, q) \in T(i_k, c_k)$ for $k = 1, 2$. Before we present the rules, let us denote a predicate msc_of, called the *most specific class of an individual*, by the following rules.

$$not\_msc\_of(X, P) \leftarrow subclass\_of(Q, P), instance\_of(X, P), instance\_of(X, Q). \tag{57}$$

$$msc\_of(X,P) \leftarrow instance\_of(X,P), not\ not\_msc\_of(X,P). \quad (58)$$

These rules state that the class p is the most specific class of an individual x if x is a member of p and x is not an instance of any subclass q of p. This will allow us to define the set $T(i_1, c_1)$ and $T(i_2, c_2)$ as follows.

$$\begin{aligned}
3\{t_1(R,P,Q), &\leftarrow comparison(X_1,C_1,Y_1,C_2), value(R,X,Y), && (59)\\
q\_d(R,P), &\quad term\_of(Y,X_1), term\_of(X,X_1), &&\\
q\_r(R,Q)\} &\quad msc\_of(X,P), msc\_of(Y,Q). &&\\
3\{t_2(R,P,Q), &\leftarrow comparison(X_1,C_1,Y_1,C_2), value(R,X,Y), && (60)\\
q\_d(R,P), &\quad term\_of(X,Y_1), term\_of(Y,Y_1), && (61)\\
q\_r(R,Q)\} &\quad msc\_of(X,P), msc\_of(Y,Q). &&
\end{aligned}$$

The following rules identify relations that are similar between c_1 and c_2:

$$\begin{aligned}
shared\_property(R) &\leftarrow comparison(X_1,C_1,Y_1,C_2), t_1(R,C_1,Q_1), t_2(R,C_2,Q_2). && (62)\\
shared\_property(S) &\leftarrow comparison(X_1,C_1,Y_1,C_2), t_1(R,P_1,C_1), t_2(R,P_2,C_2). && (63)\\
shared\_property(S) &\leftarrow comparison(X_1,C_1,Y_1,C_2), t_1(R,P,Q), t_2(R,P,Q). && (64)
\end{aligned}$$

The rules say that individuals i_1 and i_2 from class c_1 and c_2 respectively share a relation r. The first rule says that i_k $(k = 1, 2)$ is a source in the relation r (i.e., there exists some t_k such that $(r, i_k, t_k) \in \Sigma(i_k, c_k, M)$); The second rule says that i_k is a destination in the relation r (i.e., the first rule: there exists some t_k such that $(r, t_k, i_k) \in \Sigma(i_k, c_k, M)$). The third rule says that there exist some pair t_k^1, t_k^2 such that t_k^1 and t_k^2 are instances of the same class and $(r, t_k^1, t_k^2) \in \Sigma(i_k, c_k, M)$.

$$\begin{aligned}
dist\_domain(S,C_1,C_2) &\leftarrow comparison(X_1,C_1,Y_1,C_2), q\_d(S,C_1), not\ q\_d(S,C_2). && (65)\\
dist\_domain(S,C_2,C_1) &\leftarrow comparison(X_1,C_1,Y_1,C_2), q\_d(S,C_2), not\ q\_d(S,C_1). && (66)\\
dist\_range(S,C_1,C_2) &\leftarrow comparison(X_1,C_1,Y_1,C_2), q\_r(S,C_1), not\ q\_r(S,C_2). && (67)\\
dist\_range(S,C_2,C_1) &\leftarrow comparison(X_1,C_1,Y_1,C_2), q\_r(S,C_2), not\ q\_r(S,C_1). && (68)\\
dist\_property(S,P,Q,C_1,C_2) &\leftarrow comparison(X_1,C_1,Y_1,C_2), t_1(S,P,Q), not\ t_2(S,P,Q). && (69)\\
dist\_property(S,P,Q,C_2,C_1) &\leftarrow comparison(X_1,C_1,Y_1,C_2), not\ t_1(S,P,Q), t_2(S,P,Q). && (70)
\end{aligned}$$

The three predicates *dist_domain*, *dist_range*, and *dist_property* record the differences in the use of instances of c_1 and c_2 with respect to a relation r as its domain, range, or a property, similar to the three cases of similarities.
The key challenge in computing the differences/similarities between classes in `KB_Bio_101` is again the infiniteness of the grounding program.

3.4 Relationship between Individuals (Q_4)

A relationship query takes the general form of "What is the relationship between individual i_1 and individual i_2?", e.g., "what is the relationship between a `biomembrane` and a `carbohydrate`"? Since this type of query refers to a path between two individuals, it can involve significant search in the KB making it especially suitable for solution by ASP solvers. In more specific forms of this query, the choice of relationships can be limited to a specific subset of relationships in the KB. For example, "What is the structural or functional relationship between individual i_1 and individual i_2?" We can formulate this query as follows.

Given a set of literals M of an OOKB and a set of relations S, a sequence of classes alternated with relation $\omega = (c_1, s_1, c_2, s_2, \ldots, s_{n-1}, c_n)$ is called a *path between* q_1 *and* q_n with restrictive relations S in M if there exists $instance\_of(t, c_1) \in M$ and Skolem functions $f_1 = id, f_2, \ldots, f_{n-1}$ such that $value(s_i, f_i(t), f_{i+1}(t)) \in M$ for $i = 1, \ldots, n-1$ and $instance\_of(f_i(t), c_i) \in M$ for $i \geq 2$ and $s_i \in S$ for $1 \leq i < n$. A query of type $\mathbf{Q_4}$ asks for a path between c_1 and c_2 with restrictive relations in S and is encoded by the program $I(Q_4)$:

$$instance\_of(i_1, c_1). \quad instance\_of(i_2, c_2). \quad p\_relation(c_1, c_2). \quad include(r).\ (r \in S)$$

The answer to the query should indicate paths between c_1 and c_2 with restrictive relations in S. Observe that an answer can be generated by (i) selecting some atoms of the form $value(s, x, y)$ such that $s \in S$; and (ii) checking whether these atoms create a path from c_1 to c_2. We next present the set of rules $R(Q_4)$, dividing them into two groups that implement the steps (i) and (ii) as follows.

$$p\_segment(R, E, C, F, D) \leftarrow include(R), value(R, E, F), instance\_of(E, C), \quad (71)$$
$$instance\_of(F, D).$$
$$\{seg(S, E, C, F, D)\} \leftarrow p\_segment(S, E, C, F, D). \quad (72)$$
$$\leftarrow p\_relation(C_1, C_2), \{seg(\_, \_, C_1, \_, \_)\}0. \quad (73)$$
$$\leftarrow p\_relation(C_1, C_2), 2\{seg(\_, \_, C_1, \_, \_)\}. \quad (74)$$
$$\leftarrow p\_relation(C_1, C_2), \{seg(\_, \_, \_, \_, C_2)\}0. \quad (75)$$
$$\leftarrow p\_relation(C_1, C_2), 2\{seg(\_, \_, \_, \_, C_2)\}. \quad (76)$$

The first rule defines possible segments of the path. The second rule, a choice rule, picks some arbitrary segments to create the path. A segment is represented by the atom $seg(s, e, c, e', c')$ that encodes a relation s between e (an instance of class c) and e' (an instance of class c'). The rest of the rules eliminate combinations that do not create a path from c_1 to c_2. For example, the first two constraints make sure that there must be exactly one segment starting from c_1; the next two ensure that there must be exactly one segment that ends at c_2. The next four constraints make sure that the segments create a path.

$$\leftarrow p\_relation(C_1, C_2), seg(S, E, C, E_1, D), D \neq C_2, \{seg(\_, E_1, D, \_, \_)\}0. \quad (77)$$
$$\leftarrow p\_relation(C_1, C_2), seg(S, E, C, E_1, D), D \neq C_2, 2\{seg(\_, E_1, D, \_, \_)\}. \quad (78)$$
$$\leftarrow p\_relation(C_1, C_2), seg(S, E, C, E_1, D), D \neq C_2, C \neq C_1, \{seg(\_, \_, \_, E, C)\}0. \quad (79)$$
$$\leftarrow p\_relation(C_1, C_2), seg(S, E, C, E_1, D), D \neq C_2, C \neq C_1, 2\{seg(\_, \_, \_, E, C)\}. \quad (80)$$

As with other queries, the computation of an answer set of $KB(\mathbf{Q_4})$ using current ASP solvers might not be feasible.

4 Conclusions and Future Work

In this paper, we formulate the basic queries in OOKB and present ASP programs for answering these queries. We also present a practical OOKB, `KB_Bio_101`, whose size and nesserary features make the computation of the answers to these queries almost impossible using contemporary ASP solvers. Being a concrete OOKB, `KB_Bio_101` presents a real challenge for the development of ASP-solvers. This also calls for the development of novel query answering methods with huge programs in ASP.

As we have mentioned in our discussion, except for $(\mathbf{Q_1})$, the answer to $(\mathbf{Q_2})$—$(\mathbf{Q_3})$ can contain too much information that might or might not be of interested to the users. In this paper, we propose to consider only most specific classes or relation values in the answers. Our experience in the development of the system `Inquire` indicates that this might not always be a good solution. As such, it might be necessary to enrich the queries with users' preferences. This will be one of our focuses in the immediate near future.

References

1. M. Alviano, W. Faber, and N. Leone. Disjunctive asp with functions: Decidable queries and effective computation. *TPLP*, 10(4-6):497–512, 2010.
2. F. Baader, I. Horrocks, and U. Sattler. Description Logics. In *Handbook of Knowledge Representation.* Elsevier.
3. K. Barker, B. Porter, and P. Clark. A library of generic concepts for composing knowledge bases. In *Proc. 1st Int Conf on Knowledge Capture.* 14–21.
4. A. Calì, G. Gottlob, and T. Lukasiewicz. Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints. In *Database Theory - ICDT 2009.* ACM.
5. V. Chaudhri, S. Heymans, M. Wessel, and T. C. Son. Object Oriented Knowledge Bases in Logic Programming . Tech report, SRI International, 2013.
6. S. Citrigno, T. Eiter, W. Faber, G. Gottlob, C. Koch, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The dlv system: Model generator and application frontends. WLP, 128–137, 1997.
7. P. Clark and B. Porter. *KM (v2.0 and later): Users Manual*, 2011.
8. T. Eiter and M. Simkus. FDNC: Decidable nonmonotonic disjunctive logic programs with function symbols. *ACM TOCL*, 11(2), 2010.
9. M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. clasp: A conflict-driven answer set solver. LPNMR'07, LNAI 4483, 260–265. Springer-Verlag, 2007.
10. M. Gelfond and V. Lifschitz. Logic programs with classical negation. *ICLP*, 579–597, 1990.
11. D. Gunning, V. K. Chaudhri, P. Clark, K. Barker, S-Y. Chaw, M. Greaves, B. Grosof, A. Leung, D. McDonald, S. Mishra, J. Pacheco, B. Porter, A. Spaulding, D. Tecuci, and J. Tien. Project Halo Update—Progress Toward Digital Aristotle. *AI Magazine*, pages 33–58, 2010.
12. D. Magka, B. Motik, B., and I. Horrocks. Modeling Structured Domains using Description Graphs and Logic Programming. In *DL 2012.*
13. B. Motik, B. C. Grau, I. Horrocks, and U. Sattler. Representing ontologies using description logics, description graphs, and rules. AIJ, 173:1275-1309, 2009.
14. A. Overholtzer, A. Spaulding, V. K. Chaudhri, and D. Gunning. Inquire: An Intelligent Textbook. In *Proceedings of AAAI Video Competition Track*, 2012. `http://www.aaaivideos.org/2012/inquire_intelligent_textbook/`.
15. J. B. Reece, L. A. Urry, M. L. Cain, S. A. Wasserman, P. V. Minorsky, and R. B. Jackson. *Campbell Biology, 9/E.* Benjamin Cummings, 2011.
16. P. Simons, N. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1–2):181–234, 2002.
17. M. Wessel, V. Chaudhri, and S. Hyemans. Automatic Strengthening of Graph-Structured Knowledge Bases. Technical report, SRI International, 2013.