Object-Oriented Knowledge Bases in Logic Programming

VINAY K. CHAUDHRI[1], STIJN HEYMANS[1], MICHAEL WESSEL[1],
TRAN CAO SON[2]

[1] *AI Center, SRI International, Menlo Park, California, USA*
(*e-mail:* {firstname.lastname}@sri.com)
[2] *Computer Science Department, New Mexico State University, Las Cruces, New Mexico, USA*
(*e-mail:* tson@cs.nmsu.edu)

submitted 10 April 2013; accepted 23 May 2013

Abstract

It is well-known that Description Logics (DLs) that admit efficient decision procedures are unable to represent structured objects, i.e., objects whose parts are inter-connected in arbitrary instead of tree-like ways. A common solution to this problem is to extend a DL with a rule-based formalism. This either results in undecidability or requires restrictions on the shape of the rules, which typically prevent the rules from representing the required structures. In this paper, we consider an approach for modeling graph-structured objects using answer set programming. While in its full generality, reasoning with this representation is also undecidable, we consider a restriction which allows the representation of graph structured objects and yet the reasoning is decidable. We illustrate how this representation has been useful in exporting a biology knowledge base developed as part of Project Halo.

1 Introduction

As part of Project Halo (See http://www.projecthalo.com), our team at SRI has encoded a significant portion of an introductory college textbook (Reece et al. 2011) into a knowledge base called KB_Bio_101. The encoding work was done using a knowledge authoring system called AURA (Gunning et al. 2010) which uses Knowledge Machine (KM) as a knowledge representation and reasoning system (Clark and Porter 2011). Since KB_Bio_101 can be a useful resource for research on reasoning algorithms, we are interested in making it available in a way that the knowledge representation and reasoning used in it is clearly understood. In the process of developing a translation, we discovered that KB_Bio_101 cannot be directly expressed in commonly available decidable description Logics (DL) because they disallow the representation of graph-structured objects. Logic programming offers sufficient expressiveness to model the graph structures but lacks direct support for conceptual modeling primitives used in KB_Bio_101. Motivated by this problem, we consider an object-oriented language called OOKB which is an extension of answer set programming (ASP) that is capable of representing KB_Bio_101, and provides direct support for DL-style conceptual modeling primitives as well as graph structures. To give an insight into the reasoning challenges posed by OOKB, we analyze its computational properties and note that reasoning with it in full generality is undecidable. We consider syntactic restrictions under which reasoning with this representation becomes decidable. The representation features of OOKB and the syntactic restrictions considered by us are different from the

closely related prior work on $Datalog^{\pm}$ (Calì et al. 2009), $\mathbb{FDNC}$ (Eiter and Simkus 2010) programs and ASP^{fs} (Alviano et al. 2010), and none of these prior languages is adequate for capturing KB_Bio_101. More specifically, reasoning in OOKB requires rules that violate the guardedness condition of in $Datalog^{\pm}$ (see (29) and (31)—(32) in Section 4.2); and the syntactical restriction imposed in $\mathbb{FDNC}$ programs and finitely ground ASP^{fs} programs disallows the representation of graph-like structure that is needed in OOKB (see the next section).

2 Motivating Example

Suppose we wish to represent the statement: "*Every* cell *has a part a* chromosome *and a* ribosome." Given a class Living-Entity, we can represent this knowledge in a DL by:

$$Cell \sqsubseteq Living\text{-}Entity \sqcap (\exists has\text{-}part.Ribosome) \sqcap (\exists has\text{-}part.Chromosome) \tag{1}$$

Next, let us consider the following statement:

"*Every* Eukaryotic Cell *is a* Cell *and has part a* Nucleus *and a* Eukaryotic Chromosome *such that the* Eukaryotic Chromosome *is inside the* Nucleus. ".

We can capture this statement only partially using DL. Specifically, we can state:

$$\begin{aligned} Eukaryotic\text{-}Cell \sqsubseteq\ & Cell \sqcap (\exists has\text{-}part.Nucleus) \sqcap \\ & \exists (has\text{-}part.Eukaryotic\text{-}Chromosome \sqcap (\exists is\text{-}inside^{-1}.Nucleus)) \end{aligned} \tag{2}$$

The above description fails to represent that the eukaryotic chromosome is inside the same Nucleus that is the part of the Eukaryotic Cell. Indeed, expressing such knowledge would require violating the desirable tree model property (in general, the tree model property is a good indicator of decidability (Vardi 1996)). Furthermore, Eukaryotic-Cell inherits a Chromosome from its superclass Cell which is then specialized to Eukaryotic-Chromosome. The above representation does not make the relationship between the inherited Chromosome and the Eukaryotic-Chromosome defined as part of (2) explicit. Representing graph structures, and stating such relationships across a class hierarchy is crucial for giving precise answers to questions such as: What is the structure of a Eukaryotic Cell? What are the differences between a Ribosome and a Chromosome? What is the relationship between a Chromosome and a Nucleus? Such questions have been found extremely useful in the context of an education application called Inquire (Overholtzer et al. 2012).

3 Logic Programming and Answer Sets

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

$$c \leftarrow a_1, \ldots, a_m, not\ a_{m+1}, \ldots, not\ a_n \tag{3}$$

where $0 \leqslant m \leqslant n$, each a_i is a literal of a first order language and $not\ a_j$, $m < j \leqslant n$, is called a negation as failure literal (or naf-literal). A rule (program) is non-ground if it contains some variable; otherwise, it is a ground rule (program).

The Herbrand universe of a program Π is the set $\mathscr{U}_\Pi$ of terms constructable from constants and function symbols in Π. The Herbrand base of Π, $\mathscr{B}_\Pi$, is the set of ground atoms constructable using the predicate symbols in Π and the terms in $\mathscr{U}_\Pi$. A substitution δ is given by a set $\{X_1/t_1, \ldots, X_s/t_s\}$ where X_i's are distinctive variables and t_i's are terms. δ is a ground substitution if $t_i \in \mathscr{U}_\Pi$ for every i. For a literal l, $l[\delta]$ is the literal obtained from l by simultaneously replacing every occurrence of X_i by t_i for every i.

The semantics of a program is defined over ground programs. For a ground rule r of the form (3), 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 of the form (3) 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 (or a stable model)* of Π (Gelfond and Lifschitz 1990) if it satisfies the following conditions: (**i**) If Π does not contain any naf-literal then S is a 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^S.

For a non-ground program Π, a set of literals in $\mathscr{B}_\Pi$ is an answer set of Π if it is an answer set of $ground(\Pi)$ that is the set of all possible ground rules obtained from instantiating variables with terms in $\mathscr{U}_\Pi$. Π is *consistent* if $SM(\Pi) \neq \emptyset$ where $SM(\Pi)$ denotes the set of answer sets of Π. Π *entails* a ground atom a ($\Pi \models a$) if $\forall S \in SM(\Pi).[a \in S]$.

For convenience in notation, we will make use of choice atoms that are allowed to occur in a rule wherever a literal can. A choice atom is of the form $l\ S\ u$ where S is a set of literals and $l \leqslant u$ are non-negative integers; $l\ S\ u$ is true in a set of literals X if $l \leqslant |S \cap X| \leqslant u$. When $l = 0$ or $u = \infty$, they will be omitted. The set S in a choice atom $l\ S\ u$ can occur in various forms (see, e.g., (Simons et al. 2002)); e.g., it can be explicitly listed as $\{l_1,\ldots,l_n\}$ where l_i's are literals; or written in the form $\{p : q\}$ where p, q are atoms. Given a set of ground literal X, $\{p : q\} \cap X$ is the set of atoms $\{p[\delta] \mid$ there exists a ground instantiation δ such that $q[\delta] \in X\}$. Answer sets of logic programs can be computed using answer set solvers (e.g., CLASP (Gebser et al. 2007), **dlv** (Citrigno et al. 1997)).

4 Object-Oriented Knowledge Bases in ASP

We now employ the basic framework of ASP to support representation of an object-oriented knowledge base (or *OOKB*). As stated earlier, a driving motivation for this work was the need to export `KB_Bio_101` that was originally developed using Knowledge Machine (KM) (Clark and Porter 2011). The choice of terminology used in our work is influenced by the Open Knowledge Base Connectivity knowledge model (Chaudhri et al. 1998) as it had incorporated the central features of a large family of object-oriented frame-based knowledge representation systems. We will also relate the modeling primitives introduced here to their counterparts in the DL systems, but we do not make any claims about supporting the equivalent semantics.

4.1 Object-Oriented Domain

In this section, we define the vocabulary and axiom schemas necessary to define the knowledge about a domain. We consider five broad classes of knowledge: taxonomic, descriptive rules, cardinality constraints, sufficient conditions and equality statements.

4.1.1 Taxonomic Knowledge

Classes, relation declarations, and relationship between classes are specified using statements of the types (4)—(14). (4), (5), and (9) declare a class, an individual, and a relation, respectively,

(e.g., $class(cell)$ says that `cell` represents the class of cells). (6) states that c is a subclass of c' (e.g., $subclass\_of(e\_cell, cell)$ states that the class of Eukaryotic Cells (`e_cell`) is a subclass of the class of cells). (7) states that the two classes c and c' are disjoint, e.g., $disjoint(x\_chromosome, y\_chromosome)$ states that the class `x_chromosome` is disjoint from the class `y_chromosome`. (8) says that i is an individual instance of the class c.

$$class(c) \quad (4)$$
$$individual(o) \quad (5)$$
$$subclass\_of(c, c') \quad (6)$$
$$disjoint(c, c') \quad (7)$$
$$instance\_of(i, c) \quad (8)$$
$$relation(s) \quad (9)$$
$$range(s, r) \quad (10)$$
$$domain(s, d) \quad (11)$$
$$subrelation\_of(s_1, s_2) \quad (12)$$
$$compose(s_1, s_2, s_3) \quad (13)$$
$$inverse(s_1, s_2) \quad (14)$$

The domain and range of a relation are specified by (10) and (11). For example, a binary relation `has_part` whose domain and range are `tangible_entity` is represented by three atoms $relation(has\_part)$, $domain(has\_part, tangible\_entity)$, and $range(has\_part, tangible\_entity)$. (12)–(14) define the relationships between the relations of the domain. (12) states that s_1 is a *sub-relation* of s_2. (13) represents a *transfer through relation*, i.e., the composition of s_1 and s_2, $s_1 \circ s_2$, is identical to s_3. (14) indicates that s_1 is the inverse relation of s_2. An example of a sub-relation in the biology domain is $subrelation\_of(has\_functional\_part, has\_part)$: whenever $has\_functional\_part(XY)$ holds, $has\_part(X, Y)$ also holds. $compose(encloses, has\_part, encloses)$ is an example of (13): If X encloses Y and Y has Z as its part then X also encloses Z.

4.1.2 Descriptive Rules

To represent relations between individuals, we introduce atoms of the form $value(r, x, y)$ where r is a relation and x and y are terms referring to individuals. We will require that the class membership of x and y be specified if $value(r, x, y)$ is specified, e.g., to represent motivating example, we write:

$$2\{instance\_of(f_1(X), chromosome), value(has\_part, X, f_1(X))\}2 \leftarrow instance\_of(X, cell). \quad (15)$$
$$\leftarrow subclass\_of(e\_cell, cell). \quad (16)$$
$$\leftarrow subclass\_of(e\_chromo, chromosome). \quad (17)$$
$$2\{value(has\_part, X, f_2(X)), instance\_of(f_2(X), nucleus)\}2 \leftarrow instance\_of(X, e\_cell). \quad (18)$$
$$2\{value(has\_part, X, f_3(X)), instance\_of(f_3(X), e\_chromo)\}2 \leftarrow instance\_of(X, e\_cell). \quad (19)$$
$$3\{value(inside, f_3(X), f_2(X)), instance\_of(f_3(X), e\_chromo), instance\_of(f_2(X), nucleus)\}3 \leftarrow instance\_of(X, e\_cell). \quad (20)$$

Rule (15) states that for each individual X in the class `cell`, there exists $f_1(X)$ (an individual) in the class `chromosome` that is a part of X. The rule (20) states that for each X in the class `e_cell`, there exists $f_3(X)$ that is an instance of the class `e_chromo` that is inside $f_2(X)$ that is an instance of the class `nucleus`. With these rules, we are able to model the graph-structured relationship between `e_chromo`, `nucleus` and `cell`.

This leads us to define *descriptive statements* of the form

$$3\{value(s, f(X), g(X)), instance\_of(f(X), c_f), instance\_of(g(X), c_g)\}3 \leftarrow instance\_of(X, c) \quad (21)$$

where f and g are unary functions, called *Skolem functions*, such that $f \neq g$ and c is a class. f and g can be *id*, the identity function. If f (or g) is *id*, then we require that $c_f = c$ (or $c_g = c$) and we will remove the corresponding atom $instance\_of(f(X), c_f)$ and replace 3 by 2 in the head of the rule. We call $value(s, f(X), g(X))$ a *relation value literal* of c and $instance\_of(f(X), c_f)$ (or $instance\_of(g(X), c_g)$) an *instance-of literal* of c.

A descriptive statement of the form (21) describes relation values of individuals belonging to class c, represented by the atom $value(s, f(X), g(X))$: for each individual X in c, $f(X)$ (an instance of class c_f) is related to $g(X)$ (an instance of class c_g) by the relation s.

4.1.3 Cardinality Constraints on Relations

Cardinality constraints on relations are specified by statements of the following form:

$$constraint(t, f(X), s, d, n) \leftarrow instance\_of(X, c) \quad (22)$$

where s 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 s restricted on $f(X)$ has minimal (resp. maximal, exactly) n values belonging to the class d. We require that $f(X)$ must occur in a relation value literal $value(s, f(X), g(X))$ of c. For example, to state that each human cell has exactly 46 `chromosomes`, we write

$$constraint(exact, X, has\_part, chromosome, 46) \leftarrow instance\_of(X, human\_cell). \quad (23)$$

The head of (22) is called a *constraint-literal* of class c.

Observe that by setting $t = exact$, $n = 1$, $d =$`Thing`, where `Thing` is the root of the class hierarchy, a constraint of the form (22) expresses that the relation s is single-valued for all instances in the KB. When $d =$`Thing`, the constraint represents a pure number constraint. When d is some subclass of `Thing`, it represents a qualified number constraint.

4.1.4 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:

$$instance\_of(X, c) \leftarrow Body(\vec{X}) \quad (24)$$

where $Body(\vec{X})$ is a conjunction of relation value literals, instance-of literals, and constraint-literals of c, and X is a variable occurring in the body of the rule; e.g., to encode "if a `cell` has a part `nucleus`, it is a `eukaryotic cell`", we use:

$$\begin{aligned} instance\_of(X, e\_cell) \quad \leftarrow \quad & instance\_of(X_1, nucleus), \\ & relation\_value(has\_part, X, X_1), instance\_of(X, cell). \end{aligned} \quad (25)$$

4.1.5 (In)Equality Between Individual Terms

A limitation of rules (19) and (20) is that they fail to capture that $f_1(X)$ introduced as part of the definition of `cell` has been specialized to `e_chromo` as part of the rules about `e_cell`.

Since a `cell` has more than one `chromosome`, this equality cannot be inferred by simply adding cardinality constraints. To support such representation an OOKB allows a user to express equality between terms using the following rule:

$$eq(f_1(X), f_3(X)) \leftarrow instance\_of(X, e\_cell). \tag{26}$$

which says that for each individual e in the class `eukaryotic_cell`, the two terms $f_1(e)$ and $f_3(e)$ refer to the same individual. Stating such equality relationship provides a powerful modeling tool in OOKBs especially in situations where a class may inherit Skolem functions from multiple superclasses which need to refer to the same individual.

Sometimes, it is convenient also to indicate that two terms might not be equivalent. In general, an OOKB domain can contain statements for the specification of (in)equality between terms of the following form:

$$eq(t_1, t_2) \leftarrow instance\_of(X, c) \tag{27}$$

$$neq(t_1, t_2) \leftarrow instance\_of(X, c) \tag{28}$$

where c is a class, t_1 and t_2 are terms constructable from Skolem functions and the variable X.

4.1.6 Defining an OOKB Domain

Definition 1
An *OO-domain D* is a collection of rules of the form (4)–(14), (21)–(22), (24), (27), and (28).

Observe that rules of the form (7), (12), (13), (14), (21), (22), and (24) correspond to the following features in DL systems: disjointness (denoted by J), relation hierarchy (H), relation composition ($(\circ)$), inverse roles (I), existential statements (E), qualified number restrictions (Q), and sufficient properties (P), respectively. We refer to (10) and (11) as type constraints and denote them by C. A rule of type E is more general than an existential statement in a DL since it allows for the specification of non-tree structured objects.

OO-domains can be characterized by their type of rules, similar to the conventional characterization of DLs in (Schmidt-Schauß and Smolka 1991). For this purpose, we associate with each domain a label of the form Tw where w is a string over the alphabet $\{H, I, E, Q, P, C, J, (\circ)\}$. The basic part of an OO-domain is denoted with T and consists of rules of the form (4)—(6), (8) and (9). If the domain's label contains a letter in $\{H, I, E, Q, P, C, J, (\circ)\}$ then it contains rules of the corresponding form, e.g., the label $HIEP$ says that the domain contains rules for relation hierarchy, inverses, descriptive statements and sufficient conditions; etc. By a Tw-domain, we mean an OO-domain with label Tw.

4.2 Domain Independent Axioms

In this section, we will give axioms that define the meaning of various relationships that were introduced in previous section. The rules considered here can be viewed as background axioms that a reasoner would use for deriving conclusions for an OOKB domain.

4.2.1 Taxonomic Axioms

Rules (29)–(32) state transitivity of subclass relationship, inheritance of class membership, commutativity and meaning of disjointness, thus allowing for reasoning about membership of indi-

viduals with respect to a class and reasoning about relationship between classes.

$$subclass\_of(C,B) \leftarrow subclass\_of(C,A), subclass\_of(A,B). \quad (29)$$
$$instance\_of(X,C) \leftarrow instance\_of(X,D), subclass\_of(D,C). \quad (30)$$
$$disjoint(C,D) \leftarrow disjoint(D,C). \quad (31)$$
$$\neg instance\_of(X,C) \leftarrow instance\_of(X,D), disjoint(D,C). \quad (32)$$

4.2.2 Axioms for Reasoning with Relations

Rule (33) specifies the composition of two relationships. (34) and (35) specify the meaning of subrelations and inverse relations.

$$value(U,X,Z) \leftarrow compose(S,T,U), value(S,X,Y), value(T,Y,Z). \quad (33)$$
$$value(T,X,Y) \leftarrow subrelation\_of(S,T), value(S,X,Y). \quad (34)$$
$$value(T,Y,X) \leftarrow inverse(S,T), value(S,X,Y). \quad (35)$$

4.2.3 Axioms for (In)Equality Reasoning

Rules (29)–(35) allow for inheritance reasoning about class membership and relation values of an individual. In presence of rules of the form (27)-(28), a relation value can occur in different forms which represent the same relation value. For instance, given that $eq(f_1(x), f_2(x))$ holds and that both $value(has\_part, x, f_1(x))$ and $value(has\_part, x, f_2(x))$ hold, then the latter two atoms would be considered as identical. This is for example relevant when checking cardinality constraints: even though syntactically those values $f_1(x)$ and $f_2(x)$ are different, a cardinality constraint indicating that x should have strictly less than 2 parts, should still be satisfied. Thus, instead of dealing directly with *value* atoms when checking cardinality constraints we define a new $value_e$ atom which represents the set $\{value(s,x',y') \mid eq(x',x)$ and $eq(y',y)$ hold$\}$.

To support this computation, we define a predicate $substitute(x,y)$ between terms occurring in the descriptive rules of an OO-domain that indicates that x and y are identical and x could be substituted by y. The rules for propagating the equality relation and *substitute* and $value_e$ are:

$$eq(X,Y) \leftarrow eq(Y,X) \quad (36)$$
$$eq(X,Z) \leftarrow eq(X,Y), eq(Y,Z), X \neq Z \quad (37)$$
$$\leftarrow eq(X,Y), neq(X,Y) \quad (38)$$
$$\{substitute(X,Y)\} \leftarrow eq(X,Y). \quad (39)$$
$$\leftarrow eq(X,Y), \{substitute(X,Z) : eq(X,Z)\}0, \{substitute(Y,Z) : eq(Y,Z)\}0. \quad (40)$$
$$\leftarrow substitute(X,Y), substitute(X,Z), X \neq Y, X \neq Z, Y \neq Z. \quad (41)$$
$$\leftarrow substitute(X,Y), X \neq Y, neq(X,Y). \quad (42)$$
$$substitute(Y,Z) \leftarrow substitute(X,Z), X \neq Z, eq(X,Y). \quad (43)$$
$$is\_substituted(X) \leftarrow substitute(X,Y), X \neq Y. \quad (44)$$

$$substitute(X,X) \leftarrow term(X), not\ is\_substituted(X). \quad (45)$$
$$term(X) \leftarrow value(S,X,Y). \quad (46)$$
$$term(Y) \leftarrow value(S,X,Y). \quad (47)$$
$$value_e(S,X_1,Y_1) \leftarrow value(S,X,Y), substitute(X,X_1), substitute(Y,Y_1). \quad (48)$$

The first three rules express the transitivity and reflexivity of the equality between terms and that two terms cannot be specified as equal and not equal.

Rule (39) introduces a substitution for X given an *eq* statement involving X (this can be seen as a traditional *guess* step). Rules (40) ensures that that for an equality $eq(X,Y)$ there has to be at least some substitution picked for both X and Y. Rule (41) ensures that we always have at most 1 substitution and rule (43) ensures that a picked substitution does not violate any *neq* statement. We further guarantee that something is appropriately substituted by itself (rules (44)-(47)) in order to guarantee that all terms have a substitution. Rule (48) defines the predicate $value_e$ that encodes a set of relation values that are identical under the (in)equality specification. Observe that if the domain does not contain any specification of the form (27) then $value_e(s,x,y)$ holds iff $value(s,x,y)$ holds.

4.2.4 Axioms for Enforcing Constraints

To enforce the constraints (10)–(11) and (22), Π_R contains:

$$\leftarrow value(S,X,Y), domain(S,C), not\ instance\_of(X,C). \quad (49)$$
$$\leftarrow value(S,X,Y), range(S,C), not\ instance\_of(Y,C). \quad (50)$$
$$\leftarrow constraint(min,Y,S,D,M), \{value_e(S,Y,Z) : instance\_of(Z,D)\}\ M-1. \quad (51)$$

The first two rules make sure that domain and range of a relation are not violated. We note that our use of the domain and range specification is different from their conventional use in a DL (e.g., (Baader et al. 2008)) in that we do not use them for inferring the class membership of individuals. For example, suppose the domain of the relation `age` is `Living-Entity`, and that $value(age, table\text{-}25, antique)$ holds but $instance\_of(table\text{-}25, Living\text{-}Entity)$ does not, the constraint on the domain of the relation `age` is violated. This behavior is different from some DL systems which would conclude `table-25` to be an instance of `Living-Entity`.

The rule (51) enforces the minimal cardinality constraint on a set of values of a relation. Similar rules to enforce other constraints are omitted to save space. Notice that we use $value_e$ in this axiom instead of *value* because any *value* atoms that are related by *eq* need to be counted only once. Just like the domain and range constraints, we treat cardinality constraints ((22) and (51)) different from number restrictions in DLs. For example, the following OOKB constraint

$$constraint(max, X, has\_part, chromosome, 46) \leftarrow instance\_of(X, human\_cell) \quad (52)$$

means that if there would more than 46 `chromosomes` that are part of a `human cell`, there would not be an answer set (or, the KB would be inconsistent). In a DL system, in the presence of an analogous constraint, if the system encounters more than 46 `chromosome` parts of a `human cell`, it would infer that some subset of those must be equal leading to explosive case analysis. A discussion of these two different approaches to dealing with constraints can be found in (de Bruijn et al. 2005).

4.2.5 Defining a General OOKB

Definition 2 (*General OOKB*)
A general *OOKB* over a finite OO-domain D is a logic program $KB(D, D_e) = D \cup \Pi_R \cup D_e$ where (**i**) Π_R is the set of rules (29)–(35) and (36)–(51), and (**ii**) D_e is a set of ASP rules.

An OOKB $KB(D, D_e)$ is called a *taxonomical knowledge base* (or TKB) if $D_e = \emptyset$. We write $TKB(D)$ to denote $D \cup \Pi_R$. We further classify TKBs by the type of its OO-domain, e.g., if D is a $THIEQ$-domain, we say that $TKB(D)$ is a $THIEQP$ knowledge base respectively. We say that D is *consistent* if $TKB(D)$ is *consistent*.

4.3 Decidability of Reasoning

We consider decidability of the following traditional computational tasks in the context of OOKB:

- **(C_1)** *Consistency:* given an OOKB $KB(D, D_e)$, determine whether or not $KB(D, D_e)$ has an answer set.
- **(C_2)** *Concept satisfiability:* given an OOKB $KB(D, D_e)$ and an instance i of a class c, determine whether or not $KB(D, D_e) \cup \{instance\_of(i, c)\}$ has an answer set.
- **(E)** *(Ground) Entailment:* given an OOKB $KB(D, D_e)$ and a ground atom a, determine whether or not $KB(D, D_e) \models a$ holds.
- **(QA)** *Query answering:* given an OOKB $KB(D, D_e)$ and an atom q, determine all ground substitutions $\delta = \{X_1/a_1, \ldots, X_n/a_n\}$, where $\{X_1, \ldots, X_n\}$ is the set of distinct variables occurring in q, s.t. $KB(D, D_e) \models q[\delta]$.

We start off with some background on logic programming decidability results. It is well-known that deciding whether a logic program has an answer set is Σ_1^1-complete for the general first order case (Marek et al. 1992; Schlipf 1995) and entailment is undecidable (Dantsin et al. 2001): we have that (**C_1**) and (**C_2**) are Σ_1^1-complete and **(E)** is undecidable. In ASP, reasoning involves a grounding step, followed by calculating the answer sets. Therefore, the **(QA)** task is not treated explicitly in the existing ASP literature.

We next identify classes of OOKBs that yield better complexity properties for these tasks. As the complexity of general LP has been discussed extensively in the literature (e.g., (Dantsin et al. 2001; Marek et al. 1992; Schlipf 1995)), we focus on TKBs. This is because TKBs represent precisely the representation needed for `KB_Bio_101`. The first result that we obtain in this direction is related to the consistency of TKBs. Let us observe that the inconsistency of a $TKB(D)$ can arise in the different situations: (*i*) the specification of the class hierarchy is inconsistent, i.e., D contains an individual a with the specification $instance\_of(a, c_1)$ and the class relationships $disjoint(c_1, c_2)$ and $subclass\_of(c_1, c_2)$; (*ii*) the specification of (in)equality between terms is inconsistent, i.e., both $eq(x, y)$ and $neq(x, y)$ hold; or (*iii*) a domain, range, or cardinality constraint is violated, i.e., a constraint of the form (49)—(51) is violated. Since an answer set of $TKB(D)$ is an answer set of $TKB(D) \setminus \Pi_C$ where Π_C is the set of rules of the form (49)—(51), (39), (40)–(42), we can show that (**C_1**) and (**C_2**) are decidable for $THIEP(\circ)$-TKBs, i.e., (**C_1**) and (**C_2**) are decidable for TKBs without constraints.

Decidability of (**QA**) depends on whether the answer sets of TKB are finite or not. Unfortunately, finiteness of the answer set is not guaranteed as seen in the next example.

Example 1
Consider a TE OO-domain D_1 with a class a, a relation r, and the descriptive statement

$$2\{value(r, X, f(X)), instance\_of(f(X), a)\}2 \quad \leftarrow \quad instance\_of(X, a) \tag{53}$$

$$\leftarrow \quad instance\_of(c, a) \tag{54}$$

It is easy to see that $TKB(D_1)$ has an infinite answer set that can be enumerated by the function $T_{TKB(D_1)}$. The example illustrates that answer sets may be infinite in general, and thus checking ground entailment (**E**) or query answering (**QA**) is non-trivial.

We can attain decidability for ground entailment if we impose a guardedness condition. Intuitively, the application of the immediate consequence operator $T_{TKB(D)\setminus\Pi_C}$ results in larger terms (in size) if the terms appearing in the head of rules are more complex than the terms in the body. This is already the case for all rules in any TKB except possibly for sufficient conditions (24) where the head contains a variable but the body possibly more complex terms. If we can guarantee that applying $T_{TKB(D)\setminus\Pi_C}$ only results in terms of size equal or bigger than the size of the term to be proven, we can prune the search space as soon as we generate a term of size greater than the size of the term to be proven. Specifically, as each ground atom a is built over terms with a certain depth, once $T_{TKB(D)\setminus\Pi_C}$ yields atoms over terms beyond that depth and a does not appear in its result, we can be sure that a is not entailed.

Formally, for a term x in the language of $KB(D, D_e)$, $|x|$, called the size of x, is defined as the number of function symbols occurring in x. We say that an OO-domain D (and $TKB(D)$) is *guarded* if for every sufficient condition of the form (24), $1 \geqslant |Y|$ holds for term Y occurring in the body of the rule, i.e., only variables occur in rules of the form (24).

Proposition 1
(E) is decidable for TKBs with guarded and consistent $THIEPQCJ(\circ)$-domains.

The above proposition shows that entailment in OOKBs for guarded and consistent OO-domains could be verified using some ASP solvers since they provide options for limiting the maximum nesting level for complex terms (e.g., the option `maxnesting` in **dlv**); e.g., given a $TKB(D)$ and a ground atom a, by setting `maxnesting=`$|a|$, **dlv** allows us to determine whether $TKB(D) \models a$ holds.

However, guardedness still does not help with the task **(QA)** as the Herbrand universe is, in most cases, infinite and we cannot reduce **(QA)** to **(E)**. We next investigate an acyclicity condition, that leads to finite answers sets and thus decidability of **(QA)**.

Definition 3
Let D be an OO-domain and c_1 and c_2 are two classes in D. We say that a class c_1 *refers-to* a class c_2, denoted by $c_1 \prec c_2$, if **(i)** D contains a rule whose head contains some $instance\_of(Y, c_2)$ and whose body contains $instance\_of(X, c_1)$; or if **(ii)** D contains the subclass statement $subclass\_of(c_1, c_2)$.

Let $\prec^*$ be the transitive closure of $\prec$ over the set of classes in D. D is *acyclic* if there exists no class c in D s.t. $c \prec^* c$.

We say $TKB(D)$ is acyclic if D is. It is easy to check that D_1 is cyclic since $a \prec a$. We can prove:

Proposition 2
For an acyclic $THIEQCJ(\circ)$-domain D, every answer set of $TKB(D)$ is finite.

From earlier we know that every $THIEQCJ(\circ)$-TKB has at most one answer set; with Prop. 2 we then have that all the reasoning tasks ($\mathbf{C_1}$), ($\mathbf{C_2}$), **(E)**, and **(QA)** are decidable for $THIEQCJ(\circ)$ acyclic TKBs; and as a consequence, we have:

Corollary 1
Value set computation is decidable for acyclic $THIEQCJ(\circ)$ TKBs.

D_1 (Example 1) shows that Prop. 2 does not hold for cyclic OO-domains. Observe that Prop. 2 is limited to domains without sufficient conditions. This is because the body of a sufficient condition (24) often contains instance-of literal of the class occurring in the head of the rule, TKBs with sufficient conditions do not satisfy the acyclicity condition.

Observe that Prop. 2 is not a special case of Prop. 1 even though every acyclic $THIEQCJ(\circ)$-domain is guarded. Prop. 2 does not require consistency. Not all domains require sufficient conditions and sometimes, inconsistency is unavoidable.

We note that acyclicity of OOKBs is similar in spirit to *definitorial* TBoxes in DLs (Baader et al. 2008). A definitorial TBox only contains definitions and for each concept it contains only one definition that cannot refer to itself either directly or indirectly. The acyclicity conditions we have considered here does not require the KB to consist of only definitions.

5 Summary and Conclusions

The primary contributions of work reported here are in the addition of a conceptual modeling layer in the style of frame-based systems and DLs to ASP, an initial analysis of its computational properties, and its use in making available one of the largest ASP knowledge base. Given the undecidability of reasoning with this KB and its size, it poses both theoretical and practical challenges. The theoretical challenge lies in continuing to identify the weakest syntactic restrictions on the OOKB that will still allow tractable reasoning. The empirical challenge lies in identifying algorithms that support scalable reasoning with OOKBs such as `KB_Bio_101`.

References

ALVIANO, M., FABER, W., AND LEONE, N. 2010. Disjunctive asp with functions: Decidable queries and effective computation. *TPLP 10*, 4-6, 497–512.

BAADER, F., HORROCKS, I., AND SATTLER, U. 2008. Description Logics. In *Handbook of Knowledge Representation*. Elsevier.

CALÌ, A., GOTTLOB, G., AND LUKASIEWICZ, T. 2009. Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints. In *Database Theory - ICDT 2009, 12th International Conference*. ACM, St. Petersburg, Russia.

CHAUDHRI, V. K., FARQUHAR, A., FIKES, R., KARP, P. D., AND RICE, J. P. 1998. OKBC: A Programmatic Foundation for Knowledge Base Interoperability. In *Proceedings of the AAAI-98, Madison, WI*.

CITRIGNO, S., EITER, T., FABER, W., GOTTLOB, G., KOCH, C., LEONE, N., MATEIS, C., PFEIFER, G., AND SCARCELLO, F. Sep 1997. The dlv system: Model generator and application frontends. In *Proceedings of the 12th Workshop on Logic Programming WLP*, F. Bry, B. Freitag, and S. D., Eds. 128–137.

CLARK, P. AND PORTER, B. 2011. *KM (v2.0 and later): Users Manual*.

DANTSIN, E., EITER, T., GOTTLOB, G., AND VORONKOV, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Surveys 33*, 3 (Sept.), 374–425.

DE BRUIJN, J., POLLERES, A., LARA, R., AND FENSEL, D. 2005. Owl dl vs. owl flight: Conceptual modeling and reasoning for the semantic web. In *Proceedings of the 14th international conference on World Wide Web (WWW 2005)*. ACM, Chiba, Japan, 623–632.

EITER, T. AND SIMKUS, M. 2010. FDNC: Decidable nonmonotonic disjunctive logic programs with function symbols. *ACM Transactions on Computational Logic (TOCL) 11*, 2.

GEBSER, M., KAUFMANN, B., NEUMANN, A., AND SCHAUB, T. 2007. clasp: A conflict-driven answer set solver. In *Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07)*, C. Baral, G. Brewka, and J. Schlipf, Eds. Lecture Notes in Artificial Intelligence, vol. 4483. Springer-Verlag, 260–265.

GELFOND, M. AND LIFSCHITZ, V. 1990. Logic programs with classical negation. In *Logic Programming: Proceedings of the Seventh International Conference*, D. Warren and P. Szeredi, Eds. 579–597.

GUNNING, D., CHAUDHRI, V. K., CLARK, P., BARKER, K., CHAW, S.-Y., GREAVES, M., GROSOF, B., LEUNG, A., MCDONALD, D., MISHRA, S., PACHECO, J., PORTER, B., SPAULDING, A., TECUCI, D., AND TIEN, J. 2010. Project Halo Update—Progress Toward Digital Aristotle. *AI Magazine*, 33–58.

MAREK, W., NERODE, A., AND REMMEL, J. 1992. How complicated is the set of stable models of a recursive logic program? *Annals of Pure and Applied Logic 56*, 1-3, 119–135.

OVERHOLTZER, A., SPAULDING, A., CHAUDHRI, V. K., AND GUNNING, D. 2012. Inquire: An Intelligent Textbook. In *Proceedings of AAAI Video Competition Track*. `http://www.aaaivideos.org/2012/inquire_intelligent_textbook/`.

REECE, J. B., URRY, L. A., CAIN, M. L., WASSERMAN, S. A., MINORSKY, P. V., AND JACKSON, R. B. 2011. *Campbell Biology, 9/E*. Benjamin Cummings.

SCHLIPF, J. 1995. Complexity and Undecidability Results in Logic Programming. *Annals of Mathematics and Artificial Intelligence 15*, 3/4, 257–288.

SCHMIDT-SCHAUSS, M. AND SMOLKA, G. 1991. Attributive concept descriptions with complements. *Artif. Intell. 48*, 1, 1–26.

SIMONS, P., NIEMELÄ, N., AND SOININEN, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence 138*, 1–2, 181–234.

VARDI, M. Y. 1996. Why is modal logic so robustly decidable? In *Descriptive Complexity and Finite Models, Proceedings of a DIMACS Workshop, January 14-17, 1996, Princeton University*, N. Immerman and P. G. Kolaitis, Eds. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 31. American Mathematical Society, 149–184.