Object-Oriented Knowledge Bases in Logic Programming

VINAY K. CHAUDHRI¹, STIJN HEYMANS¹, MICHAEL WESSEL¹, TRAN CAO SON²

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; revised N/A; accepted N/A

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± (Calì et al. 2009), ⋄DTDNC (Eiter and Simkus 2010)
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:

\[ \text{Cell} \sqsubseteq \text{Living-Entity} \sqcap (\exists \text{has-part.Ribosome}) \sqcap (\exists \text{has-part.Chromosome}) \quad (1) \]

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:

\[ \text{Eukaryotic-Cell} \sqsubseteq \text{Cell} \sqcap (\exists \text{has-part.Nucleus}) \sqcap \exists (\text{has-part.Eukaryotic-Chromosome} \sqcap (\exists \text{is-inside}^{-1}.\text{Nucleus})) \quad (2) \]

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 \( \Pi \) is a set of rules of the form

\[ c \leftarrow a_1, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n \quad (3) \]

where \( 0 \leq m \leq n \), each \( a_i \) is a literal of a first order language and \( \text{not } a_j, m < j \leq n \), 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 \( \Pi \) is the set \( \mathbb{H}_\Pi \) of terms constructable from constants and function symbols in \( \Pi \). The Herbrand base of \( \Pi \), \( \mathcal{B}_\Pi \), is the set of ground atoms constructable using the predicate symbols in \( \Pi \) and the terms in \( \mathbb{H}_\Pi \). A substitution \( \delta \) 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. \( \delta \) is a ground substitution if \( t_i \in \mathbb{H}_\Pi \) 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 \( \text{pos}(r) = \{a_1, \ldots, a_m\} \) and \( \text{neg}(r) = \{a_{m+1}, \ldots, a_n\} \). 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) \( \text{neg}(r) \cap X \neq \emptyset \); (ii) \( \text{pos}(r) \cap X \neq \emptyset \); or (iii) \( c \in X \).

Let \( \Pi \) be a ground program. For a consistent set of ground literals \( S \), the reduct of \( \Pi \) w.r.t. \( S \), denoted by \( \Pi^S \), is the program obtained from the set of all rules of \( \Pi \) by deleting (i) each rule that
OOKBs in Logic Programming

has a naf-literal not a in its body with a ∈ 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 the least fixpoint of the immediate consequence operator, denoted by TΠ, that maps sets of ground literals into sets of ground literals; and (ii) If Π contains some naf-literal then S is an answer set of Π if S is the answer set of Π5.

For a non-ground program Π, a set of literals in BΠ 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 BΠ. Π is consistent if SM(Π) ≠ ∅ where SM(Π) denotes the set of answer sets of Π. Π entails a ground atom a (Π |= a) if ∀S ∈ SM(Π), a ∈ 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 ≤ u are non-negative integers; l S u is true in a set of literals X if l ≤ |S ∩ X| ≤ u. When l = 0 or u = ∞, 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 {l₁,...,lₙ} where lᵢ’s are literals; or written in the form {p : q} where p, q are atoms. Given a set of ground literal X, {p : q} ∩ X is the set of atoms {p[δ] | there exists a ground instantiation δ such that q[δ] ∈ 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) \quad relation(s) \quad (9)
individual(o) \quad (5) \quad range(s, r) \quad (10)
subclass_of(c, c') \quad (6) \quad domain(s, d) \quad (11)
disjoint(c, c') \quad (7) \quad subrelation_of(s_1, s_2) \quad (12)
instance_of(i, c) \quad (8) \quad 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(X,Y) 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:

$$2\{\text{instance}_{-}\text{of}(f_1(X), \text{chromosome}), \text{value}(\text{has}_{-}\text{part}, X, f_1(X))\} \leftarrow \text{instance}_{-}\text{of}(X, \text{cell}). \quad (15)$$

$$\text{subclass}_{-}\text{of}(e_{-}\text{cell}, \text{cell}). \quad (16)$$

$$2\{\text{value}(\text{has}_{-}\text{part}, X, f_2(X)), \text{instance}_{-}\text{of}(f_2(X), \text{nucleus})\} \leftarrow \text{instance}_{-}\text{of}(X, e_{-}\text{cell}). \quad (17)$$

$$2\{\text{value}(\text{has}_{-}\text{part}, X, f_3(X)), \text{instance}_{-}\text{of}(f_3(X), e_{-}\text{chromo})\} \leftarrow \text{instance}_{-}\text{of}(X, e_{-}\text{cell}). \quad (18)$$

$$3\{\text{value}(\text{inside}, f_3(X), f_2(X)), \text{instance}_{-}\text{of}(f_3(X), e_{-}\text{chromo}), \text{instance}_{-}\text{of}(f_2(X), \text{nucleus})\} \leftarrow$$

$$\text{instance}_{-}\text{of}(X, e_{-}\text{cell}). \quad (19)$$

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_chrom 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_chrom, nucleus and cell.

This leads us to define descriptive statements of the form

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

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:

$$\text{constraint}(t, f(X), s, d, n) \leftarrow \text{instance} \_ \text{of}(X, c) \tag{22}$$

where $s$ is a relation, $n$ is a non-negative integer, $d$ and $c$ are classes, and $t$ can either be $\text{min}$, $\text{max}$, or $\text{exact}$. 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 $\text{value}(s, f(X), g(X))$ of $c$. For example, to state that each human cell has exactly 46 chromosomes, we write

$$\text{constraint}(\text{exact}, X, \text{has} \_ \text{part}, \text{chromosome}, 46) \leftarrow \text{instance} \_ \text{of}(X, \text{human} \_ \text{cell}). \tag{23}$$

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

Observe that by setting $t = \text{exact}$, $n = 1$, $d = \text{Thing}$, where $\text{Thing}$ 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 $d = \text{Thing}$, the constraint represents a pure number constraint. When $d$ is some subclass of $\text{Thing}$, 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:

$$\text{instance} \_ \text{of}(X, c) \leftarrow \text{Body}(\vec{X}) \tag{24}$$

where $\text{Body}(\vec{X})$ 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:

$$\text{instance} \_ \text{of}(X, \text{e} \_ \text{cell}) \leftarrow \text{instance} \_ \text{of}(X_1, \text{nucleus}), \text{relation} \_ \text{value}(\text{has} \_ \text{part}, X, X_1), \text{instance} \_ \text{of}(X, \text{cell}). \tag{25}$$

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 $e$\_\text{chromo} as part of the rules about $e$\_\text{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:

$$\text{eq}(f_1(X), f_3(e)) \leftarrow \text{instance} \_ \text{of}(X, \text{e} \_ \text{cell}). \tag{26}$$

which says that for each individual $e$ in the class $\text{eukaryotic} \_ \text{cell}$, 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:

\[
\begin{align*}
\text{eq}(t_1, t_2) & \leftarrow \text{instance}(X, c) \\
\text{neq}(t_1, t_2) & \leftarrow \text{instance}(X, c)
\end{align*}
\]  

(27) (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 individuals with respect to a class and reasoning about relationship between classes.

\[
\begin{align*}
\text{subclass}(C, B) & \leftarrow \text{subclass}(C, A), \text{subclass}(A, B). \\
\text{instance}(X, C) & \leftarrow \text{instance}(X, D), \text{subclass}(D, C).
\end{align*}
\]  

(29) (30)

\[
\begin{align*}
\text{disjoint}(C, D) & \leftarrow \text{disjoint}(D, C). \\
\neg \text{instance}(X, C) & \leftarrow \text{instance}(X, D), \text{disjoint}(D, C).
\end{align*}
\]  

(31) (32)

#### 4.2.2 Axioms for Reasoning with Relations

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

\[
\begin{align*}
\text{value}(U, X, Z) & \leftarrow \text{compose}(S, T, U), \text{value}(S, Y), \text{value}(T, Y, Z). \\
\text{value}(T, X, Y) & \leftarrow \text{subrelation}(S, T), \text{value}(S, X, Y).
\end{align*}
\]  

(33) (34)

\[
\text{value}(T, Y, X) \leftarrow \text{inverse}(S, T), \text{value}(S, X, Y).
\]  

(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 \(value_\varepsilon\) atom which represents the set \(\{value(s, x', y') \mid eq(x', x)\text{ and } eq(y', y)\}\).

To support this computation, we define a predicate \(\text{substitute}(x, y)\) 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 \(\text{substitute}\) and \(value_\varepsilon\) are:

\[
\begin{align*}
\text{eq}(X, Y) & \leftarrow \text{eq}(Y, X) \quad (36) \\
\text{eq}(X, Z) & \leftarrow \text{eq}(X, Y), \text{eq}(Y, Z), X \neq Z \quad (37) \\
& \quad \leftarrow \text{eq}(X, Y), \text{neq}(X, Y) \quad (38) \\
\{\text{substitute}(X, Y)\} & \leftarrow \text{eq}(X, Y). \quad (39) \\
& \quad \leftarrow \text{eq}(X, Y), \{\text{substitute}(X, Z) : eq(X, Z)\}0, \{\text{substitute}(Y, Z) : eq(Y, Z)\}0. \quad (40) \\
& \quad \leftarrow \text{substitute}(X, Y), \text{substitute}(X, Z), X \neq Y, X \neq Z, Y \neq Z. \quad (41) \\
& \quad \leftarrow \text{substitute}(X, Y), X \neq Y, \text{neq}(X, Y). \quad (42) \\
\text{substitute}(Y, Z) & \leftarrow \text{substitute}(X, Z), X \neq Z, eq(X, Y). \quad (43) \\
\text{is\_substituted}(X) & \leftarrow \text{substitute}(X, Y), X \neq Y. \quad (44) \\
\text{substitute}(X, X) & \leftarrow \text{term}(X), \text{not\_is\_substituted}(X). \quad (45) \\
\text{term}(X) & \leftarrow value(S, X, Y). \quad (46) \\
\text{term}(Y) & \leftarrow value(S, X, Y). \quad (47) \\
\text{value}_\varepsilon(S, X_1, Y_1) & \leftarrow value(S, X, Y), \text{substitute}(X, X_1), \text{substitute}(Y, Y_1). \quad (48)
\end{align*}
\]

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 \(\text{neq}\) 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 \(\text{value}_\varepsilon\) 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 \(\text{value}_\varepsilon(s, x, y)\) holds iff \(\text{value}(s, x, y)\) holds.
4.2.4 Axioms for Enforcing Constraints

To enforce the constraints (10)–(11) and (22), $\Pi_R$ contains:

$$
\text{constraint}(\text{max}, X, \text{has_part}, \text{chromosome}, 46) \leftarrow \text{instance}_o f(X, \text{human}_c e l l) \tag{52}
$$

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) $\Pi_R$ 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:

(C1) **Consistency**: given an OOKB $KB(D, D_e)$, determine whether or not $KB(D, D_e)$ has an answer set.

(C2) **Concept satisfiability**: given an OOKB $KB(D, D_e)$ and an instance $i$ of a class $c$, determine whether or not $KB(D, D_e) \cup \{\text{instance}_o f(i, c)\}$ 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 $q$, such that $KB(D, D_e) \models q[\delta]$. 

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 $\Sigma_1^p$-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 $\Sigma_1^p$-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 $\text{KB}_{\text{Bio}101}$. The first result that we obtain in this direction is related to the consistency of TKBs. Let us observe that the inconsistency of a $\text{T KB}(D)$ 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 $\text{instance} \_ \text{of}(a,c_1)$ and the class relationships $\text{disjoint}(c_1,c_2)$ and $\text{subclass} \_ \text{of}(c_1,c_2)$; (ii) the specification of (in)equality between terms is inconsistent, i.e., both $eq(x,y)$ and $\text{neq}(x,y)$ 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 $\text{T KB}(D)$ is an answer set of $\text{T KB}(D) \setminus \Pi_C$ where $\Pi_C$ 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 $\text{THIEP}(\circ)$-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

\[
2\{\text{value}(r,X,f(X)),\text{instance}\_\text{of}(f(X),a)\} \leftarrow \text{instance}\_\text{of}(X,a) \quad (53)
\]

\[
\leftarrow \text{instance}\_\text{of}(c,a) \quad (54)
\]

It is easy to see that $\text{T KB}(D_1)$ has an infinite answer set that can be enumerated by the function $T_{\text{T KB}(D_1)}$. 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 $T_{\text{T KB}(\circ)\setminus \Pi_C}$ 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 $T_{\text{T KB}(\circ)\setminus \Pi_C}$ 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 $T_{\text{T KB}(\circ)\setminus \Pi_C}$ 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 $\text{KB}(D,D_e)$, $|x|$, called the size of $x$, is defined as the number of function symbols occurring in $x$. We say that an OO-domain $D$ (and $\text{T KB}(D)$) is **guarded** if for every sufficient condition of the form (24), $1 \geq |Y|$ 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 $\text{THIEP}(\circ)$-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. $c \preceq^* c$.

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 TIEQCJ($\circ$)-domain $D$, every answer set of $TKB(D)$ if it exists.

From earlier we know that every TIEQCJ($\circ$)-TKB has at most one answer set; with Prop. 2 we then have that all the reasoning tasks (C1), (C2), (E), and (QA) are decidable for TIEQCJ($\circ$) acyclic TKBs; and as a consequence, we have:

**Corollary 1**
Value set computation is decidable for acyclic TIEQCJ($\circ$) 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 TIEQCJ($\circ$)-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 OOKB Usage and Discussion

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 (Barker et al. 2001). 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. 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.

<table>
<thead>
<tr>
<th>classes</th>
<th>6430</th>
<th>subclasses_inof statements</th>
<th>6993</th>
<th>subclass_of statements</th>
<th>455</th>
<th>subrelation_inof statements</th>
<th>297</th>
<th>instance_inof statements</th>
<th>714</th>
<th>disjoint-neg statements</th>
<th>18616</th>
<th>avg. number of Skolem functions in each desc. rule</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td>individuals</td>
<td>746</td>
<td>domain constraints</td>
<td>449</td>
<td>range constraints</td>
<td>447</td>
<td>inverse relation statements</td>
<td>442</td>
<td>compose statements</td>
<td>431</td>
<td>qualified number constraints</td>
<td>936</td>
<td>descriptive rules</td>
<td>6430</td>
</tr>
<tr>
<td>relations</td>
<td>455</td>
<td></td>
<td></td>
<td>inverse relation statements</td>
<td>442</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subrelation_of statements</td>
<td>297</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>instance_of statements</td>
<td>714</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>disjoint-neg statements</td>
<td>18616</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>avg. number of Skolem functions in each desc. rule</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. Statistics on the Exported OOKB to ASP

This KB is neither guarded nor acyclic and hence the reasoning with it is likely to be undecidable. Let us consider an example that requires cyclic representation. Suppose, we wish to represent the statement: Every biomembrane has a part peripheral protein. This will be stated as:

$$2\{\text{instance}\_\text{of} (f_1(X), \text{peripheral}\_\text{protein}), \text{value}(\text{has}\_\text{part}, X, f_1(X))\} \leftarrow \text{instance}\_\text{of} (X, \text{biomembrane}).$$

(55)

Next, suppose we wish to represent “Every peripheral protein is found at the outside surface of a biomembrane”. This will be stated as:

$$2\{\text{instance}\_\text{of} (f_2(X), \text{biomembrane}), \text{value}(\text{is}\_\text{outside}, X, f_2(X))\} \leftarrow \text{instance}\_\text{of} (X, \text{peripheral}\_\text{protein}).$$

(56)

This example leads to a cycle in the representation. The biologists have been unwilling to give up expressiveness needed to model such an example. The current KB_Bio_101 has 8604 cycles.

### 6 Comparison to Related Work

Datalog$^+$ was introduced in (Call et al. 2009) as an extension of Datalog to allow existentials in the head and ensures decidability by restricting rule bodies by a guardedness condition that says that all universally quantified variables have to occur in 1 rule body atom. Allowing existentials in the head is similar conceptually to allowing function symbols in the head of rules as we do in
OOKBs. For example from (Calì et al. 2009), a Datalog± rule is
\[ \exists Y \text{supervises}(X,Y) \leftarrow \text{manager}(X). \]  
(57)

Rule (57) is clearly similar to its Skolemized version \( \text{supervises}(X,f(Y)) \leftarrow \text{manager}(X) \). The simple rule for subclass inference (29) and the disjoint rules (31) and (32) do not satisfy the definition of guardedness used in Datalog±. Similarly, the general sufficient conditions (24) cannot be expressed as the body is not necessarily guarded.

FDNC programs (see Def. 3.1 in (Eiter and Simkus 2010)) are an expressive fragment of disjunctive logic programs that include support for function symbols while still retaining decidability. They do this without an acyclicity condition that we have considered, but introduce a syntactical restriction of programs that disallows expressing graph-like structures. For example, our initial example in Section 4.1.2 requires an atom \( \text{value}(\text{inside}, f_1(X), f_2(X)) \) in the head of a rule (intuitively, enforcing a connection between nodes \( f_1(X) \) and \( f_2(X) \)). As can be seen in (Eiter and Simkus 2010), FDNC programs explicitly disallow such atoms in the head of rules. The ability to express graph-like structures has been an important requirement in the context of KB_Bio_101, and therefore, FDNC programs do not fit our needs.

Finitely ground ASPfs programs (Calimeri et al. 2008) is another class of programs with function symbols whose reasoning (skeptical or brave) remains decidable (Alviano et al. 2010). Like FDNC programs, finitely ground ASPfs programs have syntactical restriction that eliminate graph-like structures that are important for our application. For example, the rule considered in Example 1 violates the finitely ground condition in (Calimeri et al. 2008). Observe, however, that restricted classes of OOKB (e.g., guarded and consistent THIEPQCJ(◦)-domains) satisfy the finitely ground condition and thus bottom-up computations introduced in (Alviano et al. 2010) can be used in query answering for those cases.

Our work is related to attempts that extend DLs with rules to support structured objects. (Eiter et al. 2008) provides an extension of ASP with an interface to DLs. A DGLP ontology introduced in (Magka et al. 2012) is translated into a logic program whose answer set represents the semantics of the ontology. Our approach differs from this prior work in that we use ASP as a specification language and provide a set of domain independent rules for reasoning.

In summary, even though FDNC, ASPfs and Datalog± are expressive languages with overlapping features, they cannot fully capture KB_Bio_101.

7 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.


Eiter, T. and Simkus, M. 2010. FDNC: Decidable nonmonotonic disjunctive logic programs with function symbols. ACM Transactions on Computational Logic (TOCL) 11, 2.


Appendix: Sketch of Proofs

Consider a $TKB(D) = D \cup D_6 \cup \Pi_R$. Let $\Pi_C$ be the set of constrains in $\Pi_R$ (i.e., $\Pi_C$ contains the rules (49)-(51)\(^1\), (38), (40)–(42)). By definition of answer sets, we have that every answer set of $TKB(D)$ is an answer set of $\Pi = TKB(D) \setminus \Pi_C$ that satisfies $\Pi_C$.

In the following, we will prove that for a $THIEQCJ(\circ)$ acyclic domain, every answer set of $TKB(D)$ is finite if it exists. We will make use of the splitting sequence theorem in (Lifschitz and Turner 1994) to prove this conclusion. For a set of rules $X$ in $TKB(D)$, by $lit(X)$ we denote the set of literals occurring in $X$. We define the following sets of literals:

- $S_0$ is the set of literals belonging to $lit(D)$ except literals constructed from the predicate $instance_of$. In other words, $S_0$ contains
  - literals of the form (4)–(7) representing classes, individuals, subclass and disjointness relationship between classes and
  - literals of the form (9)–(14) for the declaration of relations, their cardinality constraints, and the relation hierarchy.
- $S_1$ is the set of literals constructed from atoms of the form $instance_of(x,c)$;
- $S_2$ is the set of literals constructed from atoms of the form $eq(t_1,t_2)$ and $neq(t_1,t_2)$.
- $S_3$ is the set of literals constructed from atoms of the form $value(s,t_1,t_2)$.
- $S_4$ is the set of literals constructed from atoms of the form $term(x)$.
- $S_5$ is the set of literals constructed from atoms of the form $substitute(x,y)$.
- $S_6 = lit(TKB(D)) \setminus (\bigcup_{i=0}^{5} S_i)$.

The next lemma is obvious from the definition of rules of $TKB(D)$ and the definition of a splitting sequence.

**Lemma 1**

Let $L_i = \bigcup_{j=0}^{6} S_i$ for $i = 0, \ldots, 6$. Then, $\langle L_i \rangle_{i=0}^{6}$ is a splitting sequence of $\Pi$.

The above lemma indicates that we can compute answer sets of $\Pi$ step-by-step following the splitting sequence theorem. Let us first consider the bottom of $\Pi$ with respect to $L_0$, $b_{L_0}(\Pi)$, the program consisting of all rules in $\Pi$ whose head belongs to $L_0$. In other words, $b_{L_0}(\Pi)$ consists of $D \cup \Pi'_f$ where $\Pi'_f$ consists of rules of the form (29) and (31). We have the following lemma.

**Lemma 2**

$b_{L_0}(\Pi)$ has a unique answer set $X_0$. Furthermore, if $D$ is finite then $X_0$ is finite.

**Proof.** Clearly, $b_{L_0}(\Pi)$ is a non-disjunctive program without negation-as-failure literals and thus has a unique answer set $X_0$. Finiteness of $X_0$ follows from the fact that $D \cup \Pi'_f$ does not contain function symbols and that the finiteness of $D$. \hfill \square

Observe that $X_0$ might be inconsistent. Clearly, if $X_0$ is inconsistent then $TKB(D)$ is inconsistent. Assume that $X_0$ is consistent. To apply the splitting sequence theorem, let us consider bottom

---

\(^1\) To be precise, $\Pi_C$ should contain constraints that enforce other types of constraints (e.g., $max$, and $exact$).
of $e_{L_0}(\Pi, X_0)$ with respect to $S_1 = L_1 \setminus L_0$, $b_{S_1}(e_{L_0}(\Pi, X_0))$ where $e_{L_0}(\Pi, X_0)$ is the partial evaluation of $\Pi$ with respect to $X_0$. This program consists of rules of the form (8), i.e., facts of the form

\[ \text{instance}_\text{o}_f(i, c) \]

and the rules

\[ \text{instance}_\text{o}_f(X, c_1) \leftarrow \text{instance}_\text{o}_f(X, c_2) \quad \text{(if subclass}_\text{o}_f(c_2, c_1) \in X_0) \quad (58) \]
\[ \lnot \text{instance}_\text{o}_f(X, c_1) \leftarrow \text{instance}_\text{o}_f(X, c_2) \quad \text{(if disjoint}_\text{o}_f(c_2, c_1) \in X_0) \quad (59) \]
\[ \text{instance}_\text{o}_f(h(X), c_1) \leftarrow \text{instance}_\text{o}_f(X, c_2) \quad \text{(if $h(X)$ appears in a descriptive rule of $c_2$)} \quad (60) \]
\[ \text{instance}_\text{o}_f(t_i, c_1) \leftarrow \text{instance}_\text{o}_f(X, c) \quad (61) \]

For simplicity of the notation, let $\Pi_1 = b_{S_1}(e_{L_0}(\Pi, X_0))$. We have that $\Pi_1$ does not contain the negation-as-failure operator and thus has a unique answer set $X_1$, which is the least fixpoint of the operator $T_{\Pi_1}$ that is defined as follows:

\[ T_{\Pi_1}(X) = \{ \text{head}(r) \mid \text{r} \in \text{ground}(\Pi_1) \text{ s.t. body}(r) \subseteq X \} \quad (62) \]

In other words, $X_1 = T_{\Pi_1}(\emptyset)$ where $T_{\Pi_1}(\emptyset) = \emptyset$ and $T_{\Pi_1}^{n+1}(\emptyset) = T_{\Pi_1}(T_{\Pi_1}^n(\emptyset))$ for $n \geq 0$. We have that $T_{\Pi_1}^1(\emptyset) = T_{\Pi_1}(\emptyset)$ is the collection of ground atoms of the form $\text{instance}_\text{o}_f(i, c)$ in TKB(D).

We next prove the finiteness of $X_1$ for acyclic $\text{THIEQCJ}(\circ)$-domains.

\textbf{Lemma 3}

For every acyclic $\text{THIEQCJ}(\circ)$-TKB(D), $X_1$ is finite.

\textbf{Proof.} Let $U$ be the set of positive atoms of the form $\text{instance}_\text{o}_f(t, c)$ of the program $\Pi_1$. It is easy to see that $U$ is a splitting set of $\Pi_1$ and the bottom program $\Pi'_1 = b_U(\Pi_1)$ consists of the rules (58), (60), and (61). Clearly, we have that $\Pi'_1$ has a unique answer set. Let us denote the unique answer set of $\Pi'_1$ by $U_1$. We have that $U_1 = T_{\Pi'_1}^n(\emptyset)$. We will prove next that $U_1$ is finite for acyclic $\text{THIEQCJ}(\circ)$-domain $D$.

From the construction of $\Pi'_1$ and the definition of TKB(D), we observe the following: for a class $c$ and each term $t$ constructable from Skolem functions and the constant $a$, $\text{instance}_\text{o}_f(t, c) \in T_{\Pi'_1}^{n+1}(\emptyset) \setminus T_{\Pi'_1}^n(\emptyset)$ if and only if one (or more) of the following conditions is satisfied:

1. there exists some $\text{instance}_\text{o}_f(t, c') \in T_{\Pi'_1}^n(\emptyset)$ and $\text{subclass}_\text{o}_f(c', c) \in X_0$ (because of rule (58)).
2. there exists some relation specification of the form (21) such that $t = f(t')$ or $t = g(t')$ and $\text{instance}_\text{o}_f(t', c') \in T_{\Pi'_1}^n(\emptyset)$ (because of rule (60)).

This means that if $\text{instance}_\text{o}_f(t, c) \in T_{\Pi'_1}^{n+1}(\emptyset) \setminus T_{\Pi'_1}^n(\emptyset)$ then there exists some $\text{instance}_\text{o}_f(t', c') \in T_{\Pi'_1}^n(\emptyset)$ such that $c' \prec c$. This implies that if $\text{instance}_\text{o}_f(t_k, c_k) \in U_1$ there exists a sequence of pairs $(t_1, c_1), \ldots, (t_k, c_k)$ such that $\text{instance}_\text{o}_f(t_i, c_i) \in U_1$ and $c_i \prec c_{i+1}$ for every $i = 1, \ldots, k - 1$. Because of the finiteness of the set of classes in TKB(D) and the acyclicity of TKB(D), we can conclude that $U_1$ is finite.

By the splitting theorem, we have that $X_1 = U_1 \cup V_1$ where $V_1$ is the answer set of the partial evaluation of $\Pi_1$ with respect to $U_1$ which contains the rule

\[ \lnot \text{instance}_\text{o}_f(t, c) \quad (63) \]
for instance_of(t, c') ∈ U₁ and disjoint(c, c') ∈ X₀. Clearly, we have that V₁ is finite since U₁ and X₀ are finite. The lemma is proved.

Let Π₂ = b_{S₂}(e_{₃₁}(Π₁, X₀ ∪ X₁)), we have that Π₂ contains

- a set Y₁ of facts of the form eq(t₁, t₂) or neq(t₁, t₂) where
  - eq(t₁, t₂) ∈ Y₁ iff there exists a rule of the form (27) whose head is eq(t₁, t₂) and whose body is instance_of(t', c) ∈ X₁.
  - neq(t₁, t₂) ∈ Y₁ iff there exists a rule of the form (28) whose head is neq(t₁, t₂) and whose body is instance_of(t', c) ∈ X₁.
- the following rules

  \[
  eq(X, Y) \leftarrow eq(Y, X) \tag{64} \\
  eq(X, Y) \leftarrow eq(X, Y), eq(Y, Z), X \neq Z \tag{65}
  \]

**Lemma 4**
For every acyclic THIEQCJ(∅)-TKB(D), the program Π₂ has a unique answer set X₂ that is finite.

**Proof.** Π₂ has a unique answer set because it is a program without negation-as-failure literals. The conclusion that X₂ is finite comes from the following facts:

- Y₁ is finite since X₁ is finite; and
- for every eq(x, y) ∈ X₂, x, y belongs to the set \{t, t' | eq(t, t') ∈ Y₁\} which is finite. □

Let Π₃ = b_{S₃}(e_{₃₃}(Π₁, X₀ ∪ X₁ ∪ X₂)), we have that Π₃ contains

- a set Y₂ of facts of the form value(s, t₁, t₂) where for each value(s, t₁, t₂) ∈ Y₂ there exists some instance_of(x, c) ∈ X₂ such that value(s, t₁, t₂) and instance_of(x, c) are the head and body of a rule of the form (21).
- the set of rules for propagating relations

  \[
  value(s₃, X, Z) \leftarrow value(s₁, X, Y), value(s₂, Y, Z), \tag{66} \text{(if compose}(s₁, s₂, s₃) ∈ X₀) \\
  value(s₂, X, Y) \leftarrow value(s₁, X, Y), \tag{67} \text{(if subrelation_of}(s₁, s₂) ∈ X₀) \\
  value(s₂, Y, X) \leftarrow value(s₁, X, Y), \tag{68} \text{(if inverse}(s₁, s₂) ∈ X₀)
  \]

Similar to Lemma 4, we can prove the following.

**Lemma 5**
For every acyclic THIEQCJ(∅)-TKB(D), the program Π₃ has a unique answer set X₃ that is finite.

**Proof.** Π₃ has a unique answer set because it is a non-disjunctive program without negation-as-failure literals. The conclusion that X₃ is finite comes from the following facts:

- Y₂ is finite since X₂ is finite; and
for every \( rvalue(s,x,y) \in X_3 \), \( x \) and \( y \) belongs to the set of terms \( \{ t,t' \mid value(s,t,t') \in Y_2 \} \) which is finite.

Let \( \Pi_4 = b_{S_4}(e_{L_3}(\Pi,X_0 \cup X_1 \cup X_2 \cup X_3)) \), we have that \( \Pi_4 \) contains the set of facts of the form

\[
\text{term}(x) \leftarrow (\text{if } value(s,x,y) \in X_3) \quad (69)
\]
\[
\text{term}(y) \leftarrow (\text{if } value(s,x,y) \in X_3) \quad (70)
\]

It is easy to see that the following lemma holds.

**Lemma 6**

For every acyclic \( \mathcal{THIEQCJ}(\circ)-TKB(D) \), the program \( \Pi_4 \) has a unique answer set \( X_4 \) that is finite.

**Proof.** Follows immediately from the fact that \( X_3 \) is finite (Lemma 5). \( \square \)

Let \( \Pi_5 = b_{S_5}(e_{L_4}(\Pi,\bigcup_{i=0}^4 X_i)) \), we have that \( \Pi_5 \) contains the following rules

\[
\{ \text{substitute}(x,y) \} \leftarrow (\text{if } eq(x,y) \in X_2) \quad (71)
\]
\[
\text{substitute}(y,z) \leftarrow \text{substitute}(x,z), x \neq z \quad (\text{if } eq(x,y) \in X_2) \quad (72)
\]
\[
\text{is_substituted}(x) \leftarrow \text{substitute}(x,y), x \neq y. \quad (\text{if } term(x), term(y) \in X_4) (73)
\]
\[
\text{substitute}(x,x) \leftarrow \text{not is_substituted}(x). \quad (\text{if } term(x) \in X_4) (74)
\]

The following lemma follows from the finiteness of \( X_i (i = 0,\ldots,4) \) for acyclic \( \mathcal{THIEQCJ}(\circ)-TKBs \).

**Lemma 7**

For every acyclic \( \mathcal{THIEQCJ}(\circ)-TKB(D) \), the program \( \Pi_5 \) has only finite answer sets.

**Proof.** Follows immediately from the fact that \( X_i (i = 0,\ldots,4) \) is finite that the program \( \Pi_5 \) is finite thus its answer sets are finite. \( \square \)

Lemma 2-7 and the splitting sequence theorem implies that \( X \) is an answer set of \( \Pi \) iff \( X = \bigcup_{i=0}^4 X_3 \cup X_5 \) where \( X_5 \) is an answer set of \( \Pi_5 \) and \( X_6 \) is an answer set of \( \Pi_6 = (e_{L_5}(\Pi,\bigcup_{i=0}^5 X_i)) \) which is the program consisting of rules

\[
\text{value}_c(s,x_1,y_1) \leftarrow (\text{if } \text{substitute}(x,x_1), \text{substitute}(y,y_1) \in X_5, \text{value}(s,x,y) \in X_3). \quad (75)
\]

Clearly, due to the finiteness of \( X_3 \) and \( X_5 \), we can conclude that \( \Pi_6 \) is finite. For completeness, we formulate this as the next lemma.

**Lemma 8**

For every acyclic \( \mathcal{THIEQCJ}(\circ)-TKB(D) \), the program \( \Pi_6 \) is finite and has a unique, finite answer set.

**Proof.** Follows immediately from the fact that \( X_i (i = 0,\ldots,5) \) is finite. \( \square \)

Lemma 2-8 prove Proposition 2 that proves that reasoning in acyclic \( \mathcal{THIEQCJ}(\circ)-TKB(D) \) is decidable.

We will now present the sketch of a proof for Proposition 1. As we have indicated in the main
text, given a ground atom $a$, guardedness implies that the program can be split by the set of literals $S_{a} = \{l \mid l \text{ is a literal in } TKB(D) \text{ with } |l| \leq |a| \}$ where, for every literal $x$, $|x|$ denotes the maximal size of every term occurring in $x$. The bottom part of the program $b_{S_{a}}(TKB(D))$ is finite and thus has a finite answer set. Furthermore, $b_{S_{a}}(TKB(D)) \models a$ iff $TKB(D) \models a$. This proves Proposition 1.