Abstract

Building complex knowledge based applications requires encoding large amounts of domain knowledge. After acquiring knowledge from domain experts, much of the effort in building a knowledge base goes into verifying that the knowledge is encoded correctly. We consider the problem of verifying hybrid knowledge bases that contain both Horn rules and a terminology in a description logic. Our approach to the verification problem is based on showing a close relationship to the problem of query containment. Our first contribution, based on this relationship, is presenting a thorough analysis of the decidability and complexity of the verification problem, for knowledge bases containing recursive rules and the interpreted predicates =, ≤, < and ≠. Second, we show that important new classes of constraints on correct inputs and outputs can be expressed in a hybrid setting, in which a description logic class hierarchy is also considered, and we present the first complete algorithm for verifying such hybrid knowledge bases.

Introduction

Building complex knowledge based applications requires encoding large amounts of domain knowledge. After acquiring knowledge from domain experts, much of the effort in building a knowledge base goes into verifying that the knowledge is encoded correctly. A knowledge base is verified if, for any correct set of inputs, the knowledge base will entail a correct set of outputs. Verifying a knowledge base manually is both unwieldy and unlikely to find all the possible errors in the knowledge base. Therefore, several authors have considered the problem of building tools to assist knowledge base verification. The space of verification problems varies depending on the representation language used for the knowledge base, and the way we specify constraints defining correct inputs and outputs.

This paper considers the verification problem for knowledge bases containing Horn rules and hybrid knowledge bases containing terminologies in a KL-ONE style language [Brachman and Schmolze, 1985] in addition to Horn rules. We describe novel algorithms for verifying such knowledge bases, and obtain new results concerning the complexity and decidability of the verification problem. Our algorithms also handle a wider class of input and output constraints.

We begin by showing that the verification problem is closely related to the problem of query containment, that has been studied in the database literature. As a result, we obtain algorithms for deciding the verification problem for knowledge bases containing Horn rules, and tight complexity bounds on the problem. Our results consider the cases of function-free Horn rules that may be recursive and contain the interpreted predicates =, ≤, < and ≠. In contrast, previous work has only considered non recursive Horn rule knowledge bases, and has not given any complexity or decidability results about the problem. Next, we describe the first complete algorithm for verifying hybrid knowledge bases that contain a terminology in the description logic ALCNRF [Baader and Hollunder, 1991; Buchheit et al., 1993] in addition to Horn rules.

In most works, the constraints defining correct inputs and outputs are specified also using Horn rules whose consequent is a bad predicate. An input (or output) is considered to be correct if the bad predicate is not entailed. In many domains Horn rules are not sufficient for describing correct inputs and outputs. In particular, it is often natural to express constraints using tuple generating dependencies (tgd’s) [Ullman, 1989]. In such constraints, the right hand side of the implications may also be a conjunction in which some of the variables are existentially quantified. The connection between the query containment and the verification problem also shows that verifying knowledge bases is undecidable when input or output constraints are specified using tuple generating dependencies. This is because the problem of entailment between tgd’s is undecidable.

We identify a novel class of separable tuple gener-
ating dependencies. We show that in the context of our hybrid knowledge representation language we can translate separable tgd's into Horn rules. In conjunction with our algorithm for verifying hybrid knowledge bases, we obtain a method for handling verification problems in which the input and output constraints are specified using separable tgd's. This result also entails an algorithm for query containment in the presence of separable tgd's.

**Definition of the verification problem**

Informally, a knowledge base is intended to model a space of problems. Given a problem instance, the solution to the problem will be some set of facts that are entailed by the union of the knowledge base and the problem instance. We say that the knowledge base is verified if, for any set of correct input problem instances, we will only entail correct outputs.

In our discussion we consider knowledge bases that include a set of function-free Horn rules, i.e., logical sentences of the form: \( p_1(X_1) \land \ldots \land p_n(X_n) \rightarrow q(Y) \) where \( X_1, \ldots, X_n, Y \) are tuples of variables or constants. We require that the rules are safe, i.e., a variable that appears in \( Y \) appears also in \( X_1 \) and \( \ldots \) and \( X_n \). We distinguish the set of base predicates as those predicates that do not appear in the consequents of the Horn rules. The problem instances will be specified as ground atomic facts for some of the base predicates. We also allow the interpreted predicates \( \leq, <, = \) and \( \neq \) to appear in the antecedents of the rules. Note that a ground atomic fact of the form \( p(\bar{a}) \) is also a (trivial) instance of a Horn rule.

Given a set of rules \( \mathcal{R} \) and a predicate \( P \) appearing in \( \mathcal{R} \), we define the set of rules relevant to \( P \) in \( \mathcal{R} \), denoted by \( \text{Rules}(P) \) as the minimal subset of \( \mathcal{R} \) that satisfy the following conditions:

1. If \( P \) is the predicate in the consequent of the rule \( r \), then \( r \in \text{Rules}(P) \), and
2. If the predicate \( Q \) appears in a rule \( r \in \text{Rules}(P) \), then any rule whose consequent has \( Q \) is also in \( \text{Rules}(P) \).

Given a set of rules, we can define a dependency graph, whose nodes are the predicates appearing in the rules. There is an arc from the node of predicate \( Q \) to the node of predicate \( P \) if \( Q \) appears in the antecedent of a rule whose consequent predicate is \( P \). The rules are said to be recursive if there is a cycle in the dependency graph.

When the rules are not recursive, we can unfold them. That is, obtain a logically equivalent set of rules such that the only predicates appearing in the antecedents of the rules are base predicates. It should be noted that the process of unfolding can result in an exponential number of rules. However, the exponent is only in the depth of the set of rules.

The semantics of our knowledge bases is the standard first-order logic semantics. An interpretation \( I \) of a KB \( \Delta \) contains a non-empty domain \( \mathcal{O}^I \). It assigns a \( n \)-ary relation \( \mathcal{P}^I \) over the domain \( \mathcal{O}^I \) to every \( n \)-ary predicate \( P \in \Delta \), and an element \( a^I \in \mathcal{O}^I \) to every constant \( a \in \Delta \). An interpretation \( I \) is a model of a Horn rule \( r \) if, whenever \( \alpha \) is a mapping from the variables of \( r \) to the domain \( \mathcal{O}^I \), such that \( \alpha(X_i) \in \mathcal{P}^I \) for every atom of the antecedent of \( r \), then \( \alpha(Y) \in \mathcal{P}^I \), where \( q(Y) \) is the consequent of \( r \). Finally, \( I \) is a model \( \Delta \) if it is a model of every rule in \( \Delta \).

**Correct problem instances**

A problem instance \( G \) is a set of ground atomic facts for the base predicates. Given a problem instance \( G \), the solution to the problem is the set of ground atomic facts entailed from \( \Delta \cup G \). The goal of verifying a knowledge base is to make sure that the knowledge base does not entail facts that contradict integrity constraints that are known to hold in the domain. We express such output constraints by a set of Horn rules defining a special predicate of arity 0, \( P_{\text{out}} \). We say that the output of \( \Delta \cup G \) is correct if \( \Delta \cup G \not\models P_{\text{out}} \). Similarly, we do not want to consider that all sets of ground facts are valid inputs (i.e., problem instances). Therefore, we assume that \( \Delta \) also contains a set of rules defining input constraints by a special predicate of arity 0, \( P_{\text{in}} \). A set of ground atomic facts \( G \) is a valid input if \( \Delta \cup G \not\models P_{\text{in}} \).

We can now formally define the knowledge base verification problem:

**Definition 1:** Let \( \Delta \) be a knowledge base containing the predicates \( P_{\text{in}} \) and \( P_{\text{out}} \) describing valid inputs and correct outputs. The knowledge base \( \Delta \) is said to be verified if, for any set of ground atomic facts \( G \), for which \( \Delta \cup G \not\models P_{\text{in}} \), then \( \Delta \cup G \not\models P_{\text{out}} \).

It should be noted that the verification problem is not equivalent to the unsatisfiability of the formula \( \Delta \land P_{\text{out}} \land \neg P_{\text{in}} \). In cases where all the rules are non recursive and unfolded, the verification problem can be formulated as a problem of logical entailment. In fact, the results we present in the subsequent sections can also be viewed as providing the complexity of these specialized forms of entailment.

Our definition of the verification problem differs slightly from previous definitions that were proposed in the literature (e.g., [Rousset, 1988; Ginsberg, 1988; Ginsberg and Williamson, 1993; Loiseau and Rousset, 1993]). The definition in those works did not distinguish between the predicates \( P_{\text{in}} \) and \( P_{\text{out}} \), and used a single false predicate for defining incorrect inputs and outputs. Neither formulation of the verification problem is more expressive than the other. As we see

---

1 The formula \( \Delta \land P_{\text{out}} \land \neg P_{\text{in}} \) is satisfiable if there is some model of the predicates that satisfies each rule in \( \Delta \) and \( P_{\text{out}} \) and \( \neg P_{\text{in}} \). However, the knowledge base is not verified only if there is a least fixed point model of the formula that is obtained by applying the rules to an initial set of ground facts for the base predicates.
shortly, our formulation makes the connection with the
query containment problem more explicit.

Example 1: We use the following illustrative exam-
ple throughout the paper. Consider a domain of
approving curricula for college students. The uni-
versity has two disjoint types of students, engineering
and humanities students, whose instances are described
by the unary predicates EngStud and HumStud.
Courses are either basic or advanced, described by the
predicates Basic and Adv, and they are either engi-
neering courses or humanities courses, described by
EngCourse and HumCourse. Inputs describe which
courses the student wants to take, and which courses
the student has already taken. The atom Want(s, c)
denotes that student s wants to take course c during
the current year, and Prev(s, c) denotes that s has al-
ready taken c in a previous year. The output is the
set of courses that the student will take. The atom
Take(s, c) denotes that s will take course c. The fol-
lowing rules describe our domain.

\[
\begin{align*}
    & r_1 : \text{Want}(s, c) \land \text{Qualifies}(s, c) \Rightarrow \text{Take}(s, c) \\
    & r_2 : \text{Prereq}(c_1, c_2) \land \text{Prev}(s, c_2) \Rightarrow \text{Qualifies}(s, c_1) \\
    & r_3 : \text{Year}(s, n) \land \text{Mandatory}(c, n) \Rightarrow \text{Take}(s, c)
\end{align*}
\]

Rule \( r_1 \) says that students can take a course they want if
they are qualified for it. Rule \( r_2 \) says that students are
qualified for a course if they took one of its prereq-
quisite courses. Finally, rule \( r_3 \) guarantees that students
will take the courses that are mandatory for their year.

The following is the output constraint rule stating that
humanities students cannot take advanced engineering
courses:

\[
\text{HumStud}(s) \land \text{Adv}(c) \land \text{EngCourse}(c) \land \\
\text{Take}(s, c) \Rightarrow \text{Pout}.
\]

The following two rules describe the input con-
straints specifying that engineering students are dis-
joint from humanities students, and that students do
not want to take courses they have already taken.

\[
\begin{align*}
    & r_5 : \text{EngStud}(s) \land \text{HumStud}(s) \Rightarrow \text{Pin} \\
    & r_6 : \text{Want}(s, c) \land \text{Prev}(s, c) \Rightarrow \text{Pin}
\end{align*}
\]

Our knowledge base is not verified, because we can have a valid input for which we can derive a incorrect
output. Specifically, consider the following valid input:

\{Want(S_1, C_2), HumStud(S_1), Adv(C_2), Prev(S_1, C_1),
Prereq(C_2, C_1), EngCourse(C_2)\}

The student \( S_1 \) wants to take the advanced engineering
course \( C_2 \). \( S_1 \) qualifies for the course by having taken
the prerequisite \( C_1 \). In this case, the knowledge base
would entail \( \text{Take}(S_1, C_2) \), which entails \( \text{Pout} \), i.e., the
output is incorrect.

The knowledge base designer can correct the prob-
lem by either modifying the knowledge base (e.g., refi-
n ing the rule \( r_5 \)), or by adding the (possibly very likely)
input constraint that states that humanities students
are never interested in advanced engineering courses.

Verification and the containment problem

Our approach to solving the verification problem is
based on showing a close connection to the problem of
query containment, that has been considered in the
database literature. In the next section, we show that
the relationship between these two problems yields sev-
eral new results about verifying Horn rule knowledge
bases. In particular, it provides a set of core results
about the complexity and decidability of the problem.

The query containment problem is to decide whether
in any minimal fixed-point model of the knowledge
base, the extension of one predicate contains the ex-
tension of another. Formally, given a knowledge base
\( \Delta \) and a set of ground facts \( G \), we can entail a (finite)
set of ground atomic facts for every predicate \( P \in \Delta \).
We denote by \( P^\Delta(G) \) the set of tuples \( \bar{a} \), such that
\( \Delta \cup G \models P(\bar{a}) \). If \( P \) is a proposition, i.e., a predicate
of arity 0, then \( P^\Delta(G) \) is the the set containing the
empty list if \( \Delta \cup G \models P \), and the empty set otherwise.

Definition 2: Let \( P_1 \) and \( P_2 \) be two predicates of the
same arity in the knowledge base \( \Delta \). The predicate \( P_1 \)
is contained in \( P_2 \), denoted by \( P_1 \subseteq P_2 \), if for any set of
ground atomic facts \( G \), \( P_1^\Delta(G) \subseteq P_2^\Delta(G) \).

Note that when \( P_1 \) and \( P_2 \) are propositions, the defi-

The complexity of verification

Previous work on the verification of knowledge bases
did not consider the fundamental decidability and com-
plexity of the problem. In contrast, the connection be-
tween the verification and containment problems yields
several core decidability and complexity results. This
section describes these results.

Most previous work on verification considered algo-
rithms for verifying non recursive Horn rule knowl-
edge bases without interpreted predicates. The fol-
lowing theorem establishes results about the inherent
complexity of the problem. Furthermore, the theo-
rem provides the first results concerning the verifica-
tion problem for recursive Horn rules. We assume in
our discussion that given a knowledge base \( \Delta \), when
the rules Rules(\( P_{in} \)) and Rules(\( P_{out} \)) are not recur-
sive then they are unfolded.

**Theorem 2:** Let \( \Delta \) be a Horn-rule knowledge base
without interpreted predicates. Let \( P_{in} \) and \( P_{out} \) be
predicates in \( \Delta \) describing correct inputs and outputs,
respectively.

1. If both Rules(\( P_{in} \)) and Rules(\( P_{out} \)) are not recur-
sive, then the verification problem is NP-Complete in
the size of the rules in Rules(Pin) and Rules(Pout) and polynomial in the number of rules Rules(Pin) and Rules(Pout).

2. If one of Rules(Pin) or Rules(Pout) are recursive, but not both of them, then the verification problem is complete for doubly exponential time in the size of the rules in Rules(Pin) and Rules(Pout) and polynomial in the number of rules in Rules(Pin) and Rules(Pout).

3. If both Rules(Pin) and Rules(Pout) are recursive, then the verification problem is undecidable.

The algorithm and complexity results for the first case of the theorem follow from [Sagiv and Yannakakis, 1981]. The results of the second case follow from [Chaudhuri and Vardi, 1992]. The undecidability result follows from [Shmueli, 1987].

The connection between the verification and containment also provides core complexity results for verifying Horn rule knowledge bases that include the interpreted order predicates ≤, <, = and ± in the antecedents of the rules. The following theorem provides a precise characterization of the complexity of the verification problem in this case.

**Theorem 3:** Let Δ be a Horn-rule knowledge base, possibly with the interpreted predicates ≤, <, = and ± in the antecedents of the rules. Let Pin and Pout be predicates in Δ describing correct inputs and outputs respectively.

1. If both Rules(Pin) and Rules(Pout) are not recursive, then the verification problem is \( \Pi^2 \)-complete in the size of the rules in Rules(Pin) and Rules(Pout) and the number of constants appearing in the rules. It is polynomial time in the number of rules Rules(Pin) and Rules(Pout).

2. If the rules in Rules(Pin) are recursive and Rules(Pout) are not recursive, then the verification problem is decidable and it is complete for \( \Pi^2 \) in the size of the rules in Rules(Pin) and Rules(Pout) and the number of constants appearing in them, and it is polynomial in the number of rules in Rules(Pin) and Rules(Pout).

3. If the rules in Rules(Pout) are recursive, then the verification problem is undecidable.

Note that in the above theorem there is an asymmetry between the rules defining Pin and those defining Pout. An algorithm and the upper complexity bound for the first part of the theorem follow from [Klug, 1988]. The lower bound for the first part of the theorem and the undecidability result follow from [van der Meyden, 1992]. Finally, if the rules in Rules(Pin) do not contain interpreted predicates, but the rules in Pout do contain interpreted predicates, then it follows from [Levy and Sagiv, 1995] that the verification problem is decidable also in the third case of the theorem.

**Specifying input and output constraints**

An important aspect of the knowledge base verification problem is how input and output constraints are described. In our problem definition and in most previous work in the field the constraints were specified by Horn rules defining the bad predicates Pin and Pout. However, Horn rules are not always expressive enough for describing constraints that arise in applications.

**Example 2:** Suppose we want to express the constraint on the domain of our example stating that engineering students who want to take an advanced humanities course must have previously taken a basic humanities course. Formally, we could state the constraint with the following formula which is not a Horn rule:

\[
\text{EngStud}(s) \land \text{Want}(s,c) \land \text{Adv}(c) \land \text{HumCourse}(c) \\
\Rightarrow (\exists c_1) \text{Prev}(s,c_1) \land \text{Basic}(c_1) \land \text{HumCourse}(c_1).
\]

The above example is an instance of a tuple generating dependency constraint (tdg) [Fagin, 1982; Beeri and Vardi, 1984; Yannakakis and Papadimitriou, 1980]. A tuple generating dependency constraint is a formula of the form

\[
p_1(X_1) \land \ldots \land p_m(X_n) \Rightarrow (\exists \bar{Y}) q_1(Y_1) \land \ldots \land q_m(Y_m).
\]

The tuple \( \bar{Y} \) includes the variables that appear in the right hand side and not on the left hand side. All other variables are universally quantified. Such a formula states that whenever there are facts in the knowledge base such that the conjunction on the left hand side is satisfied, then the knowledge base must also include facts such that the conjunction on the right hand side is satisfied.

An example of the usage of such constraints is to express constraints that describe test cases, which are often a natural way for an expert to describe domain constraints. That is, the expert can specify what needs to hold on the output (the right hand side of a tgd) given a certain input (the left hand side).

In [Vardi, 1984; Gurevich and Lewis, 1982] it is shown that the problem of deciding whether one tgd entails another is undecidable. Consequently, it follows that the verification problem is undecidable if we were able to express input and output constraints using arbitrary tgds.

In the next section we show how to verify hybrid knowledge bases that contain a set of extended Horn rules. Extended Horn rules contain predicates that are defined in a description logic terminology in addition to ordinary predicates. In this section we identify the class of separable tgd's, and show they can be translated to extended Horn rules whose consequents are the predicates Pin and Pout. As a result, the algorithm presented in the next section provides a method for handling verification problems in which the input and output constraints are specified by separable tgd's.

**Example 5:** We first illustrate how separable tgd's are translated to extended Horn rules using Example 2.
Informally, description logics will enable us to define the class of students that do not satisfy the right hand side of the tgd. We begin by considering the predicates Basic and HumCourse as primitive classes, and the predicate Prev as a property of objects. In a description logic we can define complete classes. The description Basic ∩ HumCourse denotes the class of objects that are basic humanities courses. We define the class C_{tgD} by the description ∀Prev.¬(Basic ∩ HumCourse) which denotes precisely the class of objects, such that fillers of the property Prev do not belong to the class Basic ∩ HumCourse. We now use the class C_{tgD} as a predicate in an extended Horn rule. Specifically, the tgd can be translated into the following rule:

$$\text{EngStud}(s) \land \text{Want}(s,c) \land \text{Adv}(c) \land \text{HumCourse}(c) \rightarrow P_m.$$  

We begin by formally defining description logic terminologies and extended rules. We then describe the algorithm for translating separable tgd's to extended Horn rules.

### Hybrid knowledge bases

A description logic is a subset of first order logic, which is especially designed to describe rich hierarchical structures. A description logic contains unary relations (called concepts) which represent classes of objects in the domain and binary relations (called roles) which describe relationships between objects. A description logic uses a set of constructors to build complex concept and role descriptions. The set of constructors varies from one language to another. In our discussion we consider the rather expressive description logic ALCN^R (which has formed the basis for the KRIS system [Baader and Hollunder, 1991]), in which descriptions can be built using the following grammar (A denotes a concept name, P_i's denote role names, C and D represent concept descriptions and R denotes a role description):

- $C, D \rightarrow A$ (primitive concept)
- $T \mid \bot$ (top, bottom)
- $C \land D \mid C \lor D$ (conjunction, disjunction)
- $¬C$ (complement)
- $\forall R.C$ (universal quantification)
- $\exists R.C$ (existential quantification)
- $(\geq n R)$ (number restrictions)
- $P_r \ldots P_m$ (role conjunction)

A terminology $T$ is a set of inclusion statements, which are of the form $C \subseteq D$, where $C$ and $D$ are concept descriptions. Intuitively, an inclusion states that every instance of the concept $C$ must be an instance of $D$. Formally, the semantics of a terminology is given via interpretations, that assign a unary relation $C^I$ to every concept name in $T$ and a binary relation $R^I$ over $O^I \times O^I$ to every role name in $T$. The extensions of concept and role descriptions are given by the following equations: ($|S|$ denotes the cardinality of a set $S$):

- $T^I = O^I, \bot^I = \emptyset, (C \land D)^I = C^I \cap D^I,$
- $(C \lor D)^I = C^I \cup D^I, (¬C)^I = O^I \setminus C^I,$
- $(\forall R.C)^I = \{d \in O^I \mid \forall e : (d,e) \in R^I \rightarrow e \in C^I\}$
- $(\exists R.C)^I = \{d \in O^I \mid \exists e : (d,e) \in R^I \land e \in C^I\}$
- $(\geq n R)^I = \{d \in O^I \mid |\{e \mid (d,e) \in R^I\}| \geq n\}$
- $(\leq n R)^I = \{d \in O^I \mid |\{e \mid (d,e) \in R^I\}| \leq n\}$
- $(P_1 \cap \ldots \cap P_m)^I = P_1^I \cap \ldots \cap P_m^I$

An interpretation $I$ is a model of $T$ if $C^I \subseteq D^I$ for every inclusion $C \subseteq D$ in $T$.

**Example 4:** In Example 3 our terminology would contain the following two inclusion statements that define precisely the concept named $C_{tgD}$.

$$C_{tgD} \subseteq \forall \text{Prev.}-\neg (\text{Basic} \cap \text{HumCourse})$$

We consider hybrid knowledge bases that contain a terminology and a set of extended Horn rules [Levy and Rousset, 1996a]. An extended Horn rule can contain in its antecedent unary and binary predicates which are concepts and roles defined in the terminology. An interpretation $I$ is a model of $\Delta$ if it is a model of both the terminology and the extended Horn rules. Algorithms for reasoning in this language are described in [Levy and Rousset, 1996a; Levy and Rousset, 1996b].

### Translating separable tgd's

Informally, the class of separable tgd's can be translated into a conjunction of concepts in ALCN^R. Formally, let $T$ be a tgd of the form $\psi \Rightarrow \phi$. Given such a conjunction $\phi$, we can define a graph $g_{\phi}$ as follows. The nodes in the graph are the variables of $\phi$ and there is an arc from a variable $X$ to a variable $Y$ if there is an atom of the form $R(X,Y)$, where $R$ is a binary predicate. A maximal path in $g_{\phi}$ is a path $X_1, \ldots, X_n$, such that there is no arc emanating from $X_n$ and no arcs coming into $X_1$. A prefix $p_1$ of a path $p$ is a subpath of $p$ that has the same initial point.

**Definition 3:** Let $T$ be a tgd of the form $\psi \Rightarrow \phi$. $T$ is a separable tgd if:

1. $\phi$ involves only unary and binary predicates,
2. $g_{\phi}$ is acyclic,
3. a variable that appears in $\psi$ can only appear in the beginning of a maximal path in $g_{\phi}$, and
4. if two maximal paths share a variable $X$, then $X$ appears only in their common prefix path.

The algorithm shown in Figure 1 creates extended Horn rules and an ALCN^R terminology that are equivalent to a separable tgd. If the given tgd describes input constraints, then the predicate in the consequent of the rules will be $P_{in}$, and otherwise, it will be $P_{out}$.
procedure tgd-to-horn(T, P)
/* T is a separable tgd of the form $\psi \Rightarrow \phi$. */
/* P is either $P_{in}$ or $P_{out}$ */

for every variable $X \in \phi$ define a concept $C_X$ as follows:
Let $C_1, \ldots, C_l$ be the literals appearing in unary atoms in $\phi$ containing $X$.
if $X$ appears only in the end of maximal paths then
$C_X = C_1 \land \ldots \land C_l$ (or $T$ if $l = 0$).
else
Let $Y_1, \ldots, Y_k$ be the variables in \{ $Y \mid R(X, Y) \in \phi$. \}
for every $Y \in \{ Y_1, \ldots, Y_k \}$ do:
Let $Role(X,Y)$ be the conjunction of the roles in the set \{ $R \mid R(X, Y) \in \phi$. \}
$C_X = (\exists Role(X,Y).C_Y) \land \ldots \land (\exists Role(Y,Z).C_Z) \land$
$C_1 \land \ldots \land C_l$.
end tgd-to-horn.

return the terminology $D_i \subseteq \neg C_{X_i}, \neg C_{X_i} \subseteq D_i$,
for $i = 1, \ldots, n$, where $X_1, \ldots, X_n$ are the variables
that appear in the beginning of maximal paths in $\phi$.
end tgd-to-horn.

Figure 1: Algorithm for translating tgd constraints

to Horn rules with a terminology.

Example 5: Considering our example tgd

$$EngStud(s) \land Want(s, c) \land Adv(c) \land HumCourse(c) \land \neg (\exists_{s_1}Prev(s, c_1) \land Basic(c_1) \land HumCourse(c_1)).$$

The right hand side of the tgd contains one maximal path $s \rightarrow c_1$. The algorithm will compute $C_{s_1} = Basic \cap HumCourse$. The concept for $s$ is $C_s = \exists_{s_1}Prev.(Basic \cap HumCourse)$. Procedure tgd-to-horn will return the terminology

$D_1 \subseteq \neg \exists_{s_1}Prev.(Basic \cap HumCourse)$
$\neg \exists_{s_1}Prev.(Basic \cap HumCourse) \subseteq D_1$,

and the rule

$$EngStud(s) \land Want(s, c) \land Adv(c) \land$$

$$HumCourse(c) \land D_1(s) \Rightarrow P_{in}.$$
that with hybrid knowledge bases we are able to handle verification problems in which the output and input constraints are expressed via the class of separable tuple generating dependencies. Such dependencies provide more expressive power than constraints that have been declaratively specified in previous work on verification.

It should be noted that our current work considers only rules whose semantics are given within first order logic. Several works have considered the verification of OPS5-style production rules (e.g., [Schmolze and Snyder, 1995], [Ginsberg and Williamson, 1993]). In such rules, the right hand side of the rules is an action that may delete facts from the knowledge base. Verification of non-recursive logical rule knowledge bases has originally been considered by Rousset [Rousset, 1988] and Ginsberg [Ginsberg, 1988], and has been extended to handle interpreted constraints ([Loiseau and Rousset, 1993] [Williamson and Dahl, 1993]).

The algorithms we presented are designed to answer the question of whether the knowledge base is verified or not. However, when the knowledge base is not verified, it is important to tell the user the possible causes of the problem, and to suggest corrective actions. The algorithms we described can be easily modified to return a counter example set of inputs in cases in which the knowledge base is not verified. Finally, we are considering extending trace-based debugging methods (as described in [Rousset and Hors, 1996]) for terminological knowledge bases to hybrid knowledge bases.

References


