Verification of Knowledge Bases based on Containment Checking

Alon Y. Levy*
Dept. of Computer Science and Engineering
University of Washington
Seattle, Washington, 98195
alon@cs.washington.edu

Marie-Christine Rousset
Dept. of Computer Science (L.R.I)
C.N.R.S & University of Paris Sud
Building 490, 91405 Orsay Cedex, France
mcr@lri.lri.fr

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. A knowledge base is verified if it can be shown that certain constraints always hold between the inputs and the outputs. We consider the knowledge base verification problem for Horn rule knowledge bases and for three kinds of constraints: I/O consistency constraints, I/O dependency constraints and Input completeness constraints. For the first two cases, we establish tight complexity results on the problem, and show in what cases it is decidable. In the third case, we show that the problem is, in general, undecidable, and we identify two decidable cases. In our analysis we show how the properties of the problem vary depending on the presence of recursion in the Horn rules, the presence of the interpreted predicates $=$, $\leq$, $<$ and $\neq$, and the presence of negation in the antecedents of the rules. Our approach to the verification problem is based on showing a close relationship to the problem of *query containment*, studied in the database literature. This connection also provides novel algorithms for the knowledge base verification problem. Finally, we provide the first algorithm for verifying hybrid knowledge bases that combine the expressive power of Horn rules and the description logic $\mathcal{ALCNR}$.

Keywords: knowledge base verification, description logics, Horn rules, database theory, query containment, hybrid languages.

*The work was done while this author was at AT&T Laboratories, Florham Park, New Jersey.

1 Introduction

Building complex knowledge based applications requires modeling and representing large amounts of knowledge. It is crucial to verify that the resulting knowledge base (KB) is correct and complete with respect to the actual knowledge that it is intended to model. Naturally, notions of correctness and completeness of a KB are impossible to capture completely by a formal definition. However, when the knowledge base is represented in a declarative logical formalism, it is possible to declaratively state various classes of constraints. This gives rise to the problem of automatically verifying these constraints, called the *knowledge base verification, validation and testing problem* (VVT). Informally, a knowledge base accepts a set of inputs (e.g., a set of ground facts in a Horn rule knowledge base). The inference mechanism of the knowledge base computes the outputs, i.e., the set of facts that can be inferred from the inputs and the content of the knowledge base. Given this view of the operation of a knowledge base, several classes of constraints can arise. For example, constraints can describe restrictions on *legal* inputs to or legal outputs from the knowledge base. Alternatively, constraints can describe *dependencies* between inputs and outputs. The problem of verifying the constraints varies depending on the representation language used in the knowledge base and on the form in which the constraints are specified.

In this paper we consider the VVT problem within a unified logical framework. We consider three classes of constraints, and therefore three instances of the VVT problem:

1. **I/O Consistency:** these constraints specify legal inputs and outputs for the knowledge base. In this case, we want to verify that whenever the inputs to the knowledge base are legal, then the outputs will be legal as well. This class of constraints has received the most attention in the VVT literature.

2. **I/O dependencies:** these constraints specify dependencies between the contents of the input and the corresponding outputs. In this case, we want to verify that these dependencies hold for any legal input to the knowledge base.

3. **Input completeness:** this class represents an especially important instance of the I/O consistency problem. In this class we specify when an input is legal by providing constraints on its completeness. That is, a constraint states that the presence of one fact in the input must imply the presence of another fact in the same input. This class of constraints is especially useful for specifying *test cases*. Testing a knowledge base w.r.t. a set of test cases is a widespread method for verifying its correctness.

We consider the VVT problem for knowledge bases specified as function-free Horn rules. Horn rule languages have formed the basis for many Artificial Intelligence applications as well as the basis for deductive and active database models. Function-free Horn rules are a natural representation language in many application domains, and are attractive because they are a tractable subset of first order logic for which several practically efficient inference procedures have been developed.

We provide novel algorithms and complexity results for the three instances of the VVT problem mentioned above.

The main tool we use to obtain our results is the connection that we establish between the VVT problem and the problem of *query containment*, that has been extensively studied in the database literature (e.g., [7, 1, 28, 16, 29, 36, 8, 32, 21, 22]). We show that viewing the VVT problem from the perspective of query containment provides a uniform view of the VVT problem which covers the different cases mentioned above. Specifically, our contributions are the following:

1. We show that for function-free Horn rule KBs, the I/O consistency and I/O dependency problems can be reformulated in terms of query containment. This connection enables us to provide the first unifying characterization of the I/O consistency and I/O dependency problems. It also provides a novel application of query containment algorithms.

2. As a result of the above connection, we obtain fundamental results on the complexity of the VVT problems, as well as novel algorithms for its solution. Our results consider the cases in which the function-free Horn rules may be recursive, may contain the interpreted predicates $=$, $\leq$, $<$ and $\neq$, and may have some limited forms of negation in the antecedents. Broadly speaking, our results show that when the Horn rules are not recursive, the VVT problems are decidable, and the results provide *tight* complexity bounds on the problem. We also show how the complexity depends on the exact form of the Horn rules. When the Horn rules are recursive, the VVT problem is undecidable. In contrast, previous work (e.g., [13, 25, 2]) provided complexity results for particular algorithms (as opposed to complexity of the problem itself). Furthermore, previous treatments were limited to the I/O consistency problem, and only for some cases of non recursive Horn rules.

3. We provide the first sound and complete algorithm for verifying *hybrid* knowledge bases, combining the expressive powers of function-free Horn rules and the description logic $\mathcal{ALCNR}$ [3, 6] (this hybrid language, CARIN, is described in [20, 19]). Description logics are useful in this context because they are especially designed to model and express constraints on domains with a rich hierarchical structure. Previous work [17, 26] provided only incomplete algorithms for verifying such knowledge bases.

4. Finally, we consider the I/O completeness problem, and show that it is related to the problem of inference of *tuple-generating dependencies* (tgd's) [34]. This relationship shows that in general, the VVT problem in the presence of I/O completeness constraints is undecidable. We identify the class of *separable* tgd's, and show that for that class it is possible to translate the tgd inference problem to the query containment problem for queries over hybrid knowledge bases. As a result, we obtain (1) a new case in which the VVT problem is decidable in the presence of I/O completeness constraints, and (2) a new case in which the inference problem of tgd's is decidable.

The paper is organized as follows. Sections 2 and 3 provide the basic definitions of the problem we consider. Section 4 establishes the relationship between the VVT and the query containment problems, and Section 5 describes the novel complexity results concerning the I/O consistency and I/O dependency problems. Section 6 introduces hybrid knowledge bases, and extends our results to this case. Section 7 considers the VVT problem in the presence of input completeness constraints, and its relationship to the problem of inferring tgd's. In section 8, our work is compared to related work and some perspectives for future work are presented.

2 Preliminaries

A knowledge base is intended to model a space of problems and their solutions. An input to a knowledge base is a set of facts which represents a particular problem instance that can be provided by a user. The corresponding output is the set of facts that are entailed by the union of the knowledge base and the given input. It represents the solution that the knowledge base provides for that problem instance.

Informally speaking, we say that a knowledge base is verified if, for any set of input facts, the input facts together with the corresponding outputs facts satisfy a set of constraints that are known to hold on the domain. We first describe the form of knowledge bases we consider in this paper.

We consider knowledge bases that include a set of function-free Horn rules, i.e., logical sentences of the form:

$$p_1(\bar{X}_1) \wedge \ldots \wedge p_n(\bar{X}_n) \Rightarrow q(\bar{Y}),$$

where $\bar{X}_1, \ldots, \bar{X}_n, \bar{Y}$ are tuples of variables or constants. We require that the rules be safe, i.e., a variable that appears in $\bar{Y}$ must also appear in $\bar{X}_1 \cup \ldots \cup \bar{X}_n$. We distinguish the set of *base predicates* as those predicates that do not appear in the consequents of the Horn rules.

Recursive rules: Given a set of rules $\mathcal{R}$, we can define a dependency graph, whose nodes are the predicates appearing in $\mathcal{R}$. In the graph, we insert 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 (as opposed to being exponential in the number of rules).

In our discussion we consider two extensions of Horn rules:

Negation on base predicates: in this case, some atoms in the antecedents are negated. We require:

- that the predicate of a negated atom be a base predicate, and

- that all the variables appearing in a negated atom appear elsewhere in a positive atom in the antecedent.

Interpreted predicates: in this case the predicates $\leq$, $<$, $=$ and $\neq$, may also occur in the antecedent of the rules. These predicates are called *interpreted predicates.* We require that the variables appearing in atoms of interpreted predicates also appear elsewhere in the antecedent in a positive atom of a non interpreted predicate. We assume that these predicates have the obvious interpretations.

All of these extensions will affect the complexity results and the corresponding algorithms.

In our discussion, we often refer to the set of rules that are *relevant* to a given predicate:

Definition 1: *Given a set of Horn rules $\mathcal{R}$ and a predicate P appearing in $\mathcal{R}$, the set of rules relevant to P in $\mathcal{R}$, denoted by $Rules(P)$ is 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 $r \in Rules(P)$*
2. *If the predicate Q appears in a rule $r \in Rules(P)$, then any rule whose consequent has Q is also in $Rules(P)$.*

The set of rules relevant to a rule r is defined to be the set of rules relevant to the predicate in the consequent of r. $\square$

Inputs and outputs: An *input* (i.e., problem instance) is specified by ground atomic facts G for some of the base predicates. The *output* of a set of rules $\mathcal{R}$, w.r.t. an input G includes the set of ground facts g such that $G \cup \mathcal{R}$ *entails* g. We define the entailment relation below.

Semantics: The semantics of our knowledge bases is given by interpretations. An *interpretation* I of a knowledge base Δ contains a non empty domain $\mathcal{O}^I$. It assigns an n-ary relation 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 $a \in \Delta$. We make the unique-names assumption, i.e., if $a \neq b$, then $a^I \neq b^I$.

An interpretation I is a model of a Horn rule r if, whenever α is a mapping from the variables of r to the domain $\mathcal{O}^I$, such that $\alpha(\bar{X}_i) \in P_i^I$ for each positive atom $P_i(\bar{X}_i)$ in the antecedent of r, and $\alpha(\bar{X}_i) \notin P_i^I$ for each negative atom $\neg P_i(\bar{X}_i)$ in the antecedent of r, then $\alpha(\bar{Y}) \in q^I$, where $q(\bar{Y})$ is the consequent of r.

An interpretation I is a model of a set of rules $\mathcal{R}$ if it is a model of every rule $r \in \mathcal{R}$.

Given a set of rules $\mathcal{R}$ and an input set of ground facts G, an atom $Q(\bar{a})$ is entailed by $\mathcal{R} \cup G$ (denoted $\mathcal{R} \cup G \models Q(\bar{a})$) if and only if $\bar{a}^I \in Q^I$ for every interpretation I that is a model of $\mathcal{R}$ and G.

Given an interpretation for the constants in $\mathcal{R} \cup G$, there is a unique model I_{min} that is the *intersection* of all models of $\mathcal{R}$ and G. It should be noted that under our definition, $\mathcal{R} \cup G \models Q(\bar{a})$ if and only if $Q(\bar{a})$ is satisfied in I_{min}. Furthermore, I_{min} can be obtained in a constructive way by successive applications of the Horn rules, starting from the ground

facts in the knowledge base, until we cannot derive any new facts. I_{min} is called the minimal fixpoint model of $\mathcal{R} \cup G$.

The notion of entailment of an atom from $\mathcal{R} \cup G$ naturally extends to any sentence $\mathcal{C}$ of first-order logic.

3 The VVT Problem

In its most general form, the VVT problem is to decide whether a set of constraints, represented by a logical sentence, is satisfied for every input to the knowledge base. Formally, this can be stated as follows.

Definition 2: *Let $\mathcal{R}$ be a set of Horn rules, and let $\mathcal{C}$ be a sentence in first-order logic. The rules $\mathcal{R}$ are verified w.r.t $\mathcal{C}$ iff for any set of input facts G, $\mathcal{R} \cup G \models \mathcal{C}$.* $\square$

In general, when the constraint $\mathcal{C}$ may be an arbitrary first-order logic sentence, it follows from the undecidability of entailment in first-order logic, that the VVT problem is also undecidable. The purpose of this paper is to investigate several classes of constraints $\mathcal{C}$ that are useful in practice and for which we show that the verification problem *is* decidable. In what follows, we describe the cases that we consider, and relate them to the general case given by Definition 2.

3.1 I/O Consistency Constraints

In the first class of constraints, we specify constraints on *legal* inputs and outputs. A knowledge base is considered to be verified if whenever the inputs are legal, then the outputs are also legal. This is the class of constraints that has received most attention in previous work in the knowledge engineering community.

Formally, consistency constraints on legal inputs and outputs are specified by Horn rules. These rules, which may be considered part of the knowledge base, define semantic inconsistency on inputs and outputs by two special predicates of arity 0, P_{in} and P_{out}. A set of input facts G is considered to be a legal input if $\mathcal{R} \cup G \not\models P_{in}$. Similarly, the corresponding output of $\mathcal{R} \cup G$ is said to be *legal* if $\mathcal{R} \cup G \not\models P_{out}$.

The VVT problem w.r.t. I/O consistency constraints is defined as follows.

Definition 3: *Let $\mathcal{R}$ be a set of Horn rules containing the predicates P_{in} and P_{out} describing constraints on legal inputs and outputs, respectively. The rules $\mathcal{R}$ are said to be verified w.r.t. P_{in} and P_{out} iff for any set of input facts G for which $\mathcal{R} \cup G \not\models P_{in}$, then $\mathcal{R} \cup G \not\models P_{out}$.* $\square$

This I/O consistency VVT problem corresponds to the instance of Definition 2, where the sentence C is $P_{out} \Rightarrow P_{in}$.

It should be noted that the verification problem is not equivalent to the unsatisfiability of the logical sentence $\mathcal{R} \wedge P_{out} \wedge \neg P_{in}$. The sentence $\mathcal{R} \wedge P_{out} \wedge \neg P_{in}$ is satisfiable if there

is *some* model that satisfies each rule in $\mathcal{R}$ and P_{out} and $\neg P_{in}$. However, the rules are not verified only if there is a *minimal fixpoint* model of $\mathcal{R} \wedge P_{out} \wedge \neg P_{in}$. In cases where all the rules are not 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.

Definition 3 differs slightly from related definitions proposed in the literature (e.g., [13, 25, 14, 23]). The definition in those works did not distinguish between the predicates P_{in} and P_{out}, and used a single *bad* predicate for defining illegal inputs and outputs. As we discuss in Section 8, previous definitions can be easily reformulated in our framework. Furthermore, our formulation makes the relationship with the query containment problem more explicit.

Example 1: We use the following illustrative example throughout the paper. Consider a domain of approving curricula for college students. The university 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 engineering 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 already 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 atom $PrereqOf(c_1, c_2)$ denotes that c_2 is a prerequisite course for c_1. The atom $Year(s, n)$ denotes that the student s is registered in the year n, and $Mandatory(c, n)$ denotes that the course c is mandatory for the year n. The following rules describe our domain.

$r_1 : Want(s, c) \wedge Qualifies(s, c) \Rightarrow Take(s, c)$
$r_2 : PrereqOf(c_1, c_2) \wedge Prev(s, c_2) \Rightarrow Qualifies(s, c_1)$
$r_3 : Year(s, n) \wedge Mandatory(c, n) \Rightarrow Take(s, c)$

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 prerequisite 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:

$r_4 : HumStud(s) \wedge Adv(c) \wedge EngCourse(c) \wedge Take(s, c) \Rightarrow P_{out}.$

The following two rules describe the input constraints specifying that engineering students are disjoint from humanities students, and that students do not want to take courses they have already taken.

$r_5 : EngStud(s) \wedge HumStud(s) \Rightarrow P_{in}$
$r_6 : Want(s, c) \wedge Prev(s, c) \Rightarrow P_{in}$

Our knowledge base is *not* verified, because we can have a legal input (w.r.t the input constraints that we consider), for which we can derive a incorrect output. Specifically, consider the following legal input:

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

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 $Take(S_1, C_2)$, which entails P_{out}, i.e., the output is incorrect.

The knowledge base designer can correct the problem by either modifying the knowledge base (e.g., refining the rule r_2), or by adding an input constraint, for example, the one stating that humanities students are never interested in advanced engineering courses. □

3.2 I/O Dependency Constraints

A second class of constraints, which is not expressible by I/O consistency constraints, includes constraints expressing dependencies which are known to exist between legal inputs and their corresponding outputs. The following example illustrates such constraints.

Example 2: Suppose we want to express the constraint on the domain of our example that students who are in their first two years and who have previously taken one advanced course must take at least one basic course. Formally we could state that constraint with the following logical formula:

$$\forall s[\exists c(Student(s) \wedge Prev(s, c) \wedge Adv(c) \wedge Year(s, n) \wedge n \leq 2) \Rightarrow \exists c(Basic(c) \wedge Take(s, c))]$$

In our framework, we formulate such a constraint by introducing two special predicates In and Out, defining the left hand side and the right hand side of the above implication, respectively. The two predicates can be defined by the following rules:

$Student(s) \wedge Prev(s, c) \wedge Adv(c) \wedge Year(s, n) \wedge n \leq 2 \Rightarrow In(s)$
$Basic(c) \wedge Take(s, c) \Rightarrow Out(s)$

The I/O dependency constraint holds if the following implication holds for every set of inputs:

$$\forall s(In(s) \Rightarrow Out(s)).$$

□

I/O dependency constraints have not been considered in previous work on the VVT problem. On the other hand, in the program verification literature (e.g. [10]), such formulations are standard. That is, they attempt to check whether for any input satisfying some preconditions, the outputs of the program satisfy certain postconditions. The definition of the VVT problem w.r.t. I/O dependency constraint is similar in spirit.

Formally, we assume that the I/O dependency constraints are specified by a set of Horn rules defining a set of pairs of predicates $(In_1, Out_1), \ldots, (In_l, Out_l)$. For every i, the predicates In_i and Out_i have the same arity. Intuitively, the constraints specify that for any input and any tuple $\bar{a}$ that is in the extension of In_i, the tuple $\bar{a}$ must also be in the extension of Out_i.

The VVT problem w.r.t. I/O dependency constraints is defined as follows.

Definition 4: *Let $\mathcal{R}$ be a set of Horn rules which includes rules defining the pairs of predicates $(In_1, Out_1), \ldots, (In_l, Out_l)$ describing I/O dependency constraints. The rules $\mathcal{R}$ are said to be verified w.r.t. the I/O dependency constraints iff, for each $i \in [1 \ldots l]$, and for any set of input facts G, if $\mathcal{R} \cup G \models In_i(\bar{a})$, then $\mathcal{R} \cup G \models Out_i(\bar{a})$.* □

This I/O dependency VVT problem corresponds to the instance of Definition 2, where the sentence C is

$$\bigwedge_{i=1}^{l} \forall \bar{X}_i (In_i(\bar{X}_i) \Rightarrow Out_i(\bar{X}_i)).$$

It should be noted that using a similar formalization, we can specify O/I dependency constraints, i.e., constraints expressing dependencies of the inputs based on the outputs.

3.3 Input Completeness Constraints

In the first class of I/O consistency constraints we specified the set of legal inputs as those for which the predicate P_{in} is not inferred. The definition of the predicate P_{in} was given by a set of Horn rules. The class of input completeness constraints enables a richer specification of the set of legal inputs. Formally, input completeness constraints are given by *tuple generating dependencies* (tgd's) [11, 4, 40], which are sentences of the form:

$$\forall \bar{X} [(\exists \bar{Z}) p_1(\bar{X}_1, \bar{Z}_1) \wedge \ldots \wedge p_n(\bar{X}_n, \bar{Z}_n) \Rightarrow (\exists \bar{Y}) q_1(\bar{X}'_1, \bar{Y}_1) \wedge \ldots \wedge q_m(\bar{X}'_m, \bar{Y}_m)].$$

The predicates $p_1, \ldots, p_n, q_1, \ldots, q_m$ are required to be base predicates, and their arguments are either variables or constants. The tuples $\bar{X}_i$ and $\bar{X}'_i$ are subsets of the tuple $\bar{X}$ and denote the variables that appear both in the left hand side and the right hand side and that are universally quantified, whereas the tuples $\bar{Z}_i$ (respectively, $\bar{Y}_i$) denote the variables that are existentially quantified in the left hand side (respectively, the right hand side). In the examples, when there is no ambiguity, we omit the universal quantifier: the variables that are common to the left hand side and the right hand side are implicitly universally quantified.

Intuitively, such a constraint specifies that if the left hand side of the sentence holds in the input, then the input must also contain facts that satisfy the right hand side.

Example 3: Suppose we want to express the constraint 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 sentence which is a tgd:

$(\exists c) EngStud(s) \wedge Want(s,c) \wedge Adv(c) \wedge HumCourse(c)$
$\Rightarrow (\exists c_1) Prev(s,c_1) \wedge Basic(c_1) \wedge HumCourse(c_1)$. $\Box$

Definition 5: *Let $\mathcal{R}$ be a set of Horn rules which includes rules:*

- *defining output constraints by a predicate P_{out}, and*
- *input constraints by a set of tgd's, Φ.*

The rules $\mathcal{R}$ are said to be verified w.r.t. input completeness constraints and output constraint iff, for any set of input facts G, if $\mathcal{R} \cup G \models \Phi$, then $\mathcal{R} \cup G \not\models P_{out}$. $\Box$

The input-completeness VVT problem corresponds to the case of Definition 2 where the sentence C is $\Phi \Rightarrow \neg P_{out}$.

4 Verification and Query Containment

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 [7, 1, 28, 16, 29, 36, 8, 32, 21, 22]. In this section we formalize the connection between the VVT problem and the query containment problem in the presence of I/O consistency and I/O dependency constraints. As a result, in Section 5 we obtain novel algorithms for solving these problems as well as the fundamental complexity results concerning it. In Section 7 we reconsider the VVT problem in the presence of input completeness constraints, and relate it to a problem of tgd entailment [11, 4, 40]. Since the tgd entailment problem is undecidable under very restrictive conditions, we identify subcases of the VVT problem that can be reformulated in terms of query containment in a hybrid language.

The query containment problem is to decide whether in any minimal fixpoint model of a set of Horn rules the extension of one predicate contains the extension of another. The problem has been extensively considered in database theory because it is an important technique for query optimization [34, 29] and related problems [21, 18, 33, 35]. Formally, given a set of Horn rules $\mathcal{R}$ and a (finite) set of ground facts G, we can entail a (finite) set of ground atomic facts for every predicate $P \in \mathcal{R}$. We denote by $P^{\mathcal{R}}(G)$ the set of tuples $\bar{a}$, such that $\mathcal{R} \cup G \models P(\bar{a})$. If P is a proposition, i.e., a predicate of arity 0, then $P^{\mathcal{R}}(G)$ is the set containing the empty list if $\mathcal{R} \cup G \models P$, and the empty set otherwise.

Definition 6: *Let $\mathcal{R}$ be a set of Horn rules, and let P_1 and P_2 be two predicates of the same arity in $\mathcal{R}$. The predicate P_1 is contained in P_2, denoted by $P_1 \subseteq P_2$, iff for any set of ground atomic facts G, $P_1^{\mathcal{R}}(G) \subseteq P_2^{\mathcal{R}}(G)$.* $\Box$

The following theorem formalizes the connection between the verification problem (w.r.t both to I/O consistency and I/O dependency constraints) and the query containment problem.

Theorem 1: *Let $\mathcal{R}$ be a set of Horn rules, possibly with negated base predicates and possibly with interpreted predicates. Suppose $\mathcal{R}$ includes:*

- *the predicate P_{in} and P_{out} defining input and output constraints, and/or*
- *the predicates $(In_1, Out_1), \ldots, (In_k, Out_k)$ defining I/O dependency constraints.*

Then,

1. *the rules $\mathcal{R}$ are verified w.r.t. the I/O consistency constraints P_{in} and P_{out} if and only the containment $P_{out} \subseteq P_{in}$ holds.*
2. *the rules $\mathcal{R}$ are verified w.r.t. the I/O dependency constraints $(In_1, Out_1), \ldots, (In_k, Out_k)$ if and only if the containment $In_i \subseteq Out_i$ holds for i, $1 \leq i \leq k$.*

□

Proof: Consider the first part of the theorem. Suppose the containment $P_{out} \subseteq P_{in}$ holds. Then for every set of ground facts G, if $\mathcal{R} \cup G \models P_{out}$ then $\mathcal{R} \cup G \models P_{in}$. Therefore, if G is a correct input (i.e., $\mathcal{R} \cup G \not\models P_{in}$), then it will only entail correct outputs (i.e., $\mathcal{R} \cup G \not\models P_{out}$).

For the other direction, suppose $\mathcal{R}$ is verified w.r.t. the I/O consistency constraints, and let G be a set of ground facts. If $\mathcal{R} \cup G \models P_{out}$, then G yields incorrect outputs. However, since $\mathcal{R}$ is verified, it means that G is not a valid input, i.e., $\mathcal{R} \cup G \models P_{in}$. Hence, $P_{out} \subseteq P_{in}$.

Consider the second part of the theorem. Suppose the containment $In_i \subseteq Out_i$ holds for every i. Then for every set of ground facts G, $In_i^{\mathcal{R}}(G) \subseteq Out_i^{\mathcal{R}}(G)$. That means that for every tuple $\bar{a}$ such that $\mathcal{R} \cup G \models In_i(\bar{a})$, then $\mathcal{R} \cup G \models Out_i(\bar{a})$. Therefore, $\mathcal{R}$ is verified w.r.t the dependency constraints defined by In_i and Out_i.

For the second direction, suppose $\mathcal{R}$ is verified w.r.t. the I/O dependency constraints In_i and Out_i. By definition, for any set of input facts G, if $\mathcal{R} \cup G \models In_i(\bar{a})$, then $\mathcal{R} \cup G \models Out_i(\bar{a})$. Therefore, $In_i^{\mathcal{R}}(G) \subseteq Out_i^{\mathcal{R}}(G)$, and In_i is contained in Out_i. □

Theorem 1 shows a direct reduction, in both directions, between the VVT problem and the problem of query containment. Therefore, we can take advantage of a collection of algorithms developed for query containment in order to address the VVT problem. In addition, the correspondence between the VVT problem and the query containment problem provides a detailed understanding of the complexity of the VVT problem. This analysis is given in the next section. It should be emphasized that previous work on the VVT problem did not consider the complexity of the problem, but only of specific algorithms.

5 The Complexity of the VVT Problem

In our complexity analysis we distinguish the case in which the Horn rules contain no interpreted predicates and no negation from the case in which they do. We assume that when the set of rules $Rules(P_{in})$ and $Rules(P_{out})$ are not recursive, then they are unfolded. The *size* of the rules in $\mathcal{R}$ refers to the maximal size of a single rule in $\mathcal{R}$. The complexity analysis for the the first case is given as follows.

Corollary 5.1: *Let $\mathcal{R}$ be a set of Horn rules without interpreted predicates or negation. Let P_{in} and P_{out} be predicates in $\mathcal{R}$ describing input and output constraints, respectively. The complexity of the VVT problem in the presence of I/O consistency constraints is the following.*

1. *If the rules $Rules(P_{out})$ are not recursive, then the verification problem is NP-Complete in the size of the rules in $Rules(P_{in})$ and $Rules(P_{out})$ and polynomial in the number of rules in $Rules(P_{in})$ and $Rules(P_{out})$.*

2. *If the rules $Rules(P_{out})$ are recursive, and the rules $Rules(P_{in})$ are not recursive, then the verification problem is complete for doubly exponential time in the size of the rules in $Rules(P_{in})$ and $Rules(P_{out})$ and polynomial in the number of rules in $Rules(P_{in})$ and $Rules(P_{out})$.*

3. *If both sets of rules $Rules(P_{in})$ and $Rules(P_{out})$ are recursive, then the verification problem is undecidable.*

□

The following provides the complexity of the VVT problem in the presence of I/O dependency constraints.

Corollary 5.2: *Let $\mathcal{R}$ be a set of Horn rules without interpreted predicates or negation. Let $(In_1, Out_1), \ldots, (In_k, Out_k)$ be predicates in $\mathcal{R}$ describing I/O dependency constraints. Let $\mathcal{R}_{Rel}$ denote the set of rules $Rules(In_1) \cup \ldots \cup Rules(In_k) \cup Rules(Out_1) \cup \ldots \cup Rules(Out_k)$. The complexity of the VVT problem in the presence of I/O dependency constraints is the following.*

1. *If the rules $Rules(In_i)$ are not recursive for $1 \leq i \leq k$, then the verification problem is NP-Complete in the size of the rules in $\mathcal{R}_{Rel}$ and polynomial in the number of rules in $\mathcal{R}_{Rel}$.*

2. *If for some i, $1 \leq i \leq k$, rules $Rules(In_i)$ are recursive, but for every i, $1 \leq i \leq k$, at most one of $Rules(In_i)$ or $Rules(Out_i)$ are recursive, then the verification problem is complete for doubly exponential time in the size of the rules in $\mathcal{R}_{Rel}$ and polynomial in the number of rules in $\mathcal{R}_{Rel}$.*

3. *If, for some i, both sets of rules In_i and Out_i are recursive, then the verification problem is undecidable.*

□

It should be noted that the above corollaries and the associated query containment algorithms provide the first complete algorithms and complexity results for the VVT problems in the presence of recursive Horn rules. Note that in all the parts of the above corollaries,

the rules in $\mathcal{R}$ that are not relevant to the consistency or dependency constraints *may* be recursive, without affecting the complexity of the VVT problem. Algorithms for the query containment problem for Horn rules without interpreted predicates and negation are given in [7, 28, 29, 8].

The algorithm and complexity results for the first case of each of the corollaries follows from [28]. The complexity results of the second case follow from [8]. The undecidability results follows from [32].

The correspondence between the VVT problem and the query containment problem also enables us to provide the first complete algorithms and complexity results for verifying Horn rule knowledge bases that include the interpreted order predicates $\leq$, $<$, $=$ and $\neq$ in the antecedents of the rules, and negation on the base predicates, and enables us to show how they differ from the simpler case of Corollaries 5.1 and 5.2. The following corollaries provide a precise characterization of the complexity of the verification problem in this case.

Corollary 5.3: *Let $\mathcal{R}$ be a set of Horn rules, possibly with the interpreted predicates $\leq$, $<$, $=$ and $\neq$ and negation. Let P_{in} and P_{out} be predicates in $\mathcal{R}$ defining I/O consistency constraints, respectively. The complexity of the VVT problem in the presence of I/O consistency constraints is the following.*

1. *If both sets of rules $Rules(P_{in})$ and $Rules(P_{out})$ are not recursive, then the verification problem is Π_2^P-Complete in the size of the rules in $Rules(P_{in})$ and $Rules(P_{out})$. The complexity is polynomial in the number of rules in $Rules(P_{in})$ and $Rules(P_{out})$.*

2. *If the rules in $Rules(P_{in})$ are recursive and $Rules(P_{out})$ are not recursive, then the verification problem is decidable and it is complete for Π_2^P in the size of the rules in $Rules(P_{in})$ and $Rules(P_{out})$. The complexity is polynomial in the number of rules in $Rules(P_{in})$ and $Rules(P_{out})$.*

3. *If the rules in $Rules(P_{out})$ are recursive, then the verification problem is undecidable.*

□

The following is the analogous result for the I/O dependency problem.

Corollary 5.4: *Let $\mathcal{R}$ be a set of Horn rules, possibly with the interpreted predicates $\leq$, $<$, $=$ and $\neq$ and negation. Let $(In_1, Out_1), \ldots, (In_k, Out_k)$ be predicates in $\mathcal{R}$ describing I/O dependency constraints. Let $\mathcal{R}_{Rel}$ denote the set of rules $Rules(In_1) \cup \ldots \cup Rules(In_k) \cup Rules(Out_1) \cup \ldots \cup Rules(Out_k)$. The complexity of the VVT problem in the presence of I/O dependency constraints is the following.*

1. *If all the rules in $\mathcal{R}_{Rel}$ are not recursive, then the verification problem is Π_2^P-Complete in the size of the rules in $\mathcal{R}_{Rel}$ and polynomial in the number of rules in $\mathcal{R}_{Rel}$.*

2. *If the rules* $Rules(In_1), \ldots, Rules(In_k)$ *are not recursive, but some of the rules in* $Rules(Out_1), \ldots, Rules(Out_k)$ *are recursive, then the verification problem is decidable and it is complete for* Π_2^P *in the size of the rules in* $\mathcal{R}_{Rel}$*, and polynomial in the number of rules in* $\mathcal{R}_{Rel}$*.*

3. *If some of the rules in* $Rules(In_1), \ldots, Rules(In_k)$ *are recursive, then the problem is undecidable.*

□

It is important to note that in the above corollaries there is an asymmetry between the rules defining P_{in} (In_i) and those defining P_{out} (Out_i) (which follows from the analogous asymmetry in the analysis of the query containment problem). An algorithm and the upper complexity bound for the first part of Corollaries 5.3 and 5.4 follow from [16]. The algorithm and upper bound complexity result for the second cases is given in [21]. The lower bound for the first part of the corollaries and the undecidability result follow from [37]. Finally, we note that the VVT problem considered here would remain decidable also if the rules have function symbols, as long as the rules are not recursive. However, if we allow negation on predicates other than base predicates, then the VVT problem is undecidable, even when the rules are not recursive.

Negation and Input Completeness Constraints: In our discussion we have considered cases in which the Horn rules contain negated base predicates in their antecedents. Except for providing additional modeling power as a representation language, negation can also be used for expressing certain kinds of input completeness constraints. The following example illustrates such a usage.

Example 4: Suppose we want to express the following input completeness constraint on the domain of our example: for second-year students, all the courses that they have taken previously were mandatory courses. This constraint can be specified by the following sentence:

$$Prev(s, c) \wedge Year(s, 2) \Rightarrow Mandatory(c, 1).$$

Note that in this example, *Mandatory* is a base predicate, and therefore the constraint specifies a condition on the completeness of the input.

The constraint, specified in this form, is a special case of a tuple-generating dependency. However, using negation on base predicates, this sentence can be translated to the following Horn rule defining P_{in}:

$$Prev(s, c) \wedge Year(s, 2) \wedge \neg Mandatory(c, 1) \Rightarrow P_{in}.$$

As a result, verifying the set of rules in the presence of such input completeness constraints can be done using the techniques described in this section for the VVT problem in the presence of I/O consistency constraints. It is easy to see that this transformation can be

done for any tuple generating dependency that does not contain existential variables on the right-hand side. Obviously, an analogous transformation can be done for certain kinds of O/I dependency constraints. □

6 Verifying Hybrid Knowledge Bases

Horn rule languages are well suited to capture fine-grained relational knowledge but they are not expressive enough to model complex structural knowledge. In contrast, description logics are a family of representation languages that have been designed especially to model rich hierarchies of classes of objects. Several applications, such as combining information from multiple heterogeneous sources, modeling complex physical devices, significantly benefit from combining the expressive power of both formalisms. In this section we consider hybrid knowledge bases using the CARIN family of languages, which was designed to extend Horn rules with the expressive power of description logics.

We show that the correspondence between the VVT problem and the query containment problem also enables us to provide the first sound and complete algorithm for verifying hybrid knowledge bases.

Aside from being a more expressive language for domain modeling, the expressive power of CARIN provides two other advantages in the context of the VVT problem:

- Description logics provide a natural way to express constraints on predicates appearing in the Horn rules, such as disjointness and subsumption between predicates.
- As we show in the next section, the added expressive power of CARIN enables us to express a class of input completeness constraints, and therefore to solve the VVT problem in the presence of that class of constraints.

We begin by introducing the syntax and semantics of the CARIN languages.

6.1 CARIN Knowledge Bases

CARIN is a family of languages, each of which combines a description logic $\mathcal{L}$ with Horn rules. We denote a specific language in CARIN by CARIN-$\mathcal{L}$. A set of rules in CARIN-$\mathcal{L}$ contains two components, the first is a description-logic terminology, and the second is a set of function-free *extended* Horn rules. The terminology is a set of statements in $\mathcal{L}$ about concepts and roles in the domain. Extended Horn rules are rules in which concept and role descriptions can appear as predicate names in the antecedents. Predicate names appearing in the Horn rules that do not appear in the terminology are called *ordinary predicates*. Ordinary predicates can be of any arity. In this paper we consider the language CARIN-$\mathcal{ALCNR}$. We briefly review the description logic $\mathcal{ALCNR}$ [3, 6] below.

6.1.1 The Description Logic $\mathcal{ALCNR}$

A description logic contains unary relations (called concepts) which represent sets of objects in the domain and binary relations (called roles) which describe relationships between objects. Expressions in the terminology are built from concept and role names and from concept and role *descriptions*, which denote complex concepts and roles. Descriptions in $\mathcal{ALCNR}$ are defined using the following syntax (A denotes a primitive concept name, P_i's denote primitive role names, C and D represent concept descriptions and R denotes a role description):

$C, D \rightarrow$	$A \mid$	(primitive concept)
	$\top \mid \bot \mid$	(top, bottom)
	$C \sqcap D \mid C \sqcup D \mid$	(conjunction, disjunction)
	$\neg C \mid$	(complement)
	$\forall R.C \mid$	(universal quantification)
	$\exists R.C \mid$	(existential quantification)
	$(\geq\ n\, R) \mid (\leq\ n\, R)$	(number restrictions)
$R \rightarrow$	$P_1 \sqcap \ldots \sqcap P_m$	(role conjunction)

The sentences in a terminological component of a knowledge base are either *concept inclusions* or *role definitions*. A concept inclusion is of the form $C \sqsubseteq D$, where C and D are concept descriptions. Intuitively an inclusion states that every instance of the concept C must be an instance of D. A role definition is a formula of the form $P := R$, where P is a role name and R is a role description.

Semantics of $\mathcal{ALCNR}$: The semantics of a terminology $\mathcal{T}$ is given via *interpretations*. An interpretation I contains a non empty domain $\mathcal{O}^I$. It assigns a unary relation C^I to every concept in $\mathcal{T}$, and a binary relation R^I over $\mathcal{O}^I \times \mathcal{O}^I$ to every role R in $\mathcal{T}$. The extensions of concept and role descriptions are given by the following equations: ($\sharp S$ denotes the cardinality of a set S):

$$\top^I = \mathcal{O}^I$$
$$\bot^I = \emptyset$$
$$(C \sqcap D)^I = C^I \cap D^I$$
$$(C \sqcup D)^I = C^I \cup D^I$$
$$(\neg C)^I = \mathcal{O}^I \setminus C^I$$
$$(\forall R.C)^I = \{d \in \mathcal{O}^I \mid \forall e : (d,e) \in R^I \rightarrow e \in C^I\}$$
$$(\exists R.C)^I = \{d \in \mathcal{O}^I \mid \exists e : (d,e) \in R^I \wedge e \in C^I\}$$
$$(\geq\ n\, R)^I = \{d \in \mathcal{O}^I \mid \sharp\{e \mid (d,e) \in R^I\} \geq n\}$$
$$(\leq\ n\, R)^I = \{d \in \mathcal{O}^I \mid \sharp\{e \mid (d,e) \in R^I\} \leq n\}$$
$$(P_1 \sqcap \ldots \sqcap P_m)^I = P_1^I \cap \ldots \cap P_m^I$$

An interpretation I is a model of a terminology $\mathcal{T}$ if $C^I \subseteq D^I$ for every inclusion $C \sqsubseteq D$ in the terminology, and $P^I = R^I$ for every role definition $P := R$. We say that C *is subsumed by* D w.r.t. $\mathcal{T}$ if $C^I \subseteq D^I$ in every model I of $\mathcal{T}$.

6.1.2 Semantics of CARIN:

The semantics of a set of extended Horn rules is defined in exactly the same way as in Section 2. The only subtle point to note is that we always consider atoms of concept predicates to be positive atoms. For example, the atom $\neg A(x)$ is considered to be a positive atom whose predicate name is $\neg A$, which is a concept in $\mathcal{ALCNR}$. We do not allow negated atoms of roles in the Horn rules.

Given a set of extended Horn rules $\mathcal{R}$, and a terminology $\mathcal{T}$, we define entailment as follows. Given a set of ground facts G for the ordinary predicates, the concepts and the roles, and given a query of the form $Q(\bar{a})$, where Q can be any ordinary predicate, concept or role, we say that $\mathcal{R} \cup \mathcal{T} \cup G \models Q(\bar{a})$ if $\bar{a}^I \in Q^I$ for every interpretation I such that:

- I is a model of $\mathcal{R} \cup \mathcal{T}$, and
- for every atom $P(\bar{b}) \in G$, then $\bar{b}^I \in P^I$, and
- for every ordinary base predicate E and tuple $\bar{b}$, $\bar{b}^I \in E^I$ only if $E(\bar{b}) \in G$.

Sound and complete entailment algorithms for CARIN-$\mathcal{ALCNR}$ are given in [19, 20]. Note that when the Horn rules are recursive, the entailment problem for CARIN-$\mathcal{ALCNR}$ is not decidable [20].

The following example illustrates the use of CARIN for expressing more complex I/O dependency constraints.

Example 5: Suppose we want to express an I/O dependency constraint stating that all the students of a given year who have previously taken only basic courses, have to take an advanced course. Using predicates In and Out, it can be stated by the two following sentences. Note that the first sentence cannot be expressed as a Horn rule, even with negation on base predicates:

$Student(s) \wedge \forall c[Prev(s,c) \Rightarrow Basic(c)] \Rightarrow In(s)$
$Adv(c) \wedge Take(s,c) \Rightarrow Out(s)$

In CARIN, we express this constraint by defining the following terminology and extended rule:

$C \sqsubseteq \forall Prev.Basic$
$\forall Prev.Basic \sqsubseteq C$

$Student(s) \wedge C(s) \Rightarrow In(s)$. □

In [19] we describe an algorithm for query containment for non recursive CARIN-$\mathcal{ALCNR}$ rules. That algorithm entails the following result.

Corollary 6.1: *Let $\mathcal{R}$ be a set of extended Horn rules in* CARIN-$\mathcal{ALCNR}$ *without interpreted predicates, and $\mathcal{T}$ be a terminology in $\mathcal{ALCNR}$. Assume that the magnitude of the integers*

used in the number restrictions in $\mathcal{T}$ is bounded by the size of $\mathcal{T}$. Assume that $\mathcal{R}$ includes rules defining the predicates P_{in} and P_{out}, describing I/O consistency constraints, respectively.

If both sets of rules $Rules(P_{in})$ and $Rules(P_{out})$ are not recursive then the VVT problem w.r.t. I/O consistency constraints is decidable in time that is doubly exponential in the size of the rules in $Rules(P_{in})$ and $Rules(P_{out})$ and the size of $\mathcal{T}$, and is polynomial in the number of rules in $Rules(P_{in})$ and $Rules(P_{out})$. □

A similar corollary can be stated for the VVT problem w.r.t. I/O dependency constraints.

7 VVT in the Presence of Input Completeness Constraints

In this section we consider the VVT problem in the presence of input completeness constraints. Unfortunately, since the entailment problem of tgd's is undecidable [38, 15], it follows immediately that the VVT problem in the presence of input completeness and I/O completeness constraints is undecidable. In this section we identify the class of *separable* tgds, for which we show that the problem *is* decidable. The key to obtaining our result is an algorithm for translating a set of separable tgd's into a set of extended Horn rules in CARIN, and therefore obtaining a reduction of the the VVT problem in the presence of input completeness constraints to the VVT problem in the presence of I/O consistency constraints in CARIN. The following is an example of our method.

Example 6: Suppose we want to express the input completeness constraint stating that engineering students, who want to take an advanced humanities course, *must* have previously taken a basic humanities course. Formally, this constraint can be stated using the following tgd:

$$EngStud(s) \wedge Want(s,c) \wedge Adv(c) \wedge HumCourse(c) \\ \Rightarrow (\exists c_1) Prev(s,c_1) \wedge Basic(c_1) \wedge HumCourse(c_1).$$

The idea behind the translation is to create a concept description that describes the set 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 in a terminology, and the predicate $Prev$ as a role. The description $Basic \sqcap HumCourse$ denotes the class of objects that are basic humanities courses. The class C_{tgd} can be defined by the description $\forall Prev.\neg(Basic \sqcap HumCourse)$ which denotes precisely the class of objects, such that all fillers of the role $Prev$ do not belong to the class $Basic \sqcap HumCourse$. We now use the class C_{tgd} as a predicate in an extended Horn rule.

The result of our translation would be the terminology containing the following two inclusion statements:

$$C_{tgd} \sqsubseteq \forall Prev.\neg(Basic \sqcap HumCourse)$$
$$\forall Prev.\neg(Basic \sqcap HumCourse) \sqsubseteq C_{tgd}$$

The tgd would be translated into the following extended Horn rule, defining the predicate P_{in}:

$$EngStud(s) \wedge Want(s,c) \wedge Adv(c) \wedge HumCourse(c) \wedge C_{tgd}(s) \Rightarrow P_{in}. \ \Box$$

In what follows, we formally define the class of separable tgd's and then describe the transformation algorithm.

Separable TGD's

Suppose T is a tgd of the form $\psi \Rightarrow \phi$. Given the sentence ϕ, we can define a graph g_ϕ as follows. The nodes in the graph are the variables of ϕ, 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_ϕ 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 7: *Let T be a tgd of the form $\psi \Rightarrow \phi$, such that ϕ mentions only unary and binary predicates. T is a separable tgd iff:*

1. *g_ϕ is acyclic,*
2. *a variable that appears in ψ can only appear in the beginning of a maximal path in g_ϕ,*
3. *all the variables in ϕ that appear in the beginning of a maximal path also appear in ψ, and*
4. *if two maximal paths in g_ϕ share a variable X, then X appears only in their common prefix.*

$\Box$

The intuition behind Definition 7 is that the right-hand side of a separable TGD (i.e., the formula ϕ) can be equivalently rewritten as a conjunction of *cr-formulas*, defined as follows:

Definition 8: *A formula f is a cr-formula on the variable X if it has the following form:*

$$C_1(X) \wedge \ldots \wedge C_n(X) \wedge \exists\, Y_1, \ldots, Y_m [R_1(X,Y_1) \wedge f_1(Y_1) \wedge \ldots \wedge R_m(X,Y_m) \wedge f_m(Y_m)]$$

where $C_1, \ldots, C_n$ are concepts, $R_1, \ldots, R_m$ are roles and $f_1(Y_1), \ldots, f_m(Y_m)$ are cr-formulas on $Y_1, \ldots, Y_m$, respectively. Note that either m or n may be 0. $\Box$

Observation 2: Let T be a separable tgd of the form $\psi \Rightarrow \phi$, such that the variables common to ϕ and ψ are $X_1, \ldots, X_n$. Then, ϕ is logically equivalent to a sentence of the form $f_1 \wedge \ldots \wedge f_n$, such that for all i:

1. f_i is a cr-formula on X_i,
2. X_i appears only in f_i, and
3. if $i \neq j$, then f_i and f_j do not share any variables.

$\Box$

The Transformation Algorithm

In Figure 1 we show the algorithm for transforming a given separable TGD. The output of the algorithm is a terminology and a set of extended Horn rules defining P_{in}, i.e., defining an input consistency constraint.

procedure tgd-to-horn(T)
/* T is a separable tgd of the form $\psi \Rightarrow \phi$. */

/* The algorithm returns a set of extended Horn rules and a terminology. */

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 ϕ containing X.
if X appears only in the end of maximal paths **then** $C_X = C_1 \sqcap \ldots \sqcap C_l$ (or $\top$ 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 $\{R \mid R(X,Y) \in \phi\}$.
$C_X = (\exists\ Role_{X,Y_1}.C_{Y_1}) \sqcap \ldots \sqcap (\exists\ Role_{X,Y_k}.C_{Y_k}) \sqcap C_1 \sqcap \ldots \sqcap C_l$.

return the terminology $D_i \sqsubseteq \neg C_{X_i}$, $\neg C_{X_i} \sqsubseteq D_i$, and the rules $\psi \wedge D_i(X_i) \Rightarrow P_{in}$,
where $X_1, \ldots, X_n$ are the variables that appear in the beginning of maximal paths in ϕ.

end tgd-to-horn.

Figure 1: Algorithm for translating a set of separable tgd's to a set of extended Horn rules with a terminology.

Example 7: Considering our example tgd

$$EngStud(s) \wedge Want(s,c) \wedge Adv(c) \wedge HumCourse(c) \Rightarrow (\exists c_1) Prev(s,c_1) \wedge Basic(c_1) \wedge HumCourse(c_1).$$

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

$D_1 \sqsubseteq \neg\exists\ Prev.(Basic \sqcap HumCourse)$
$\neg\exists\ Prev.(Basic \sqcap HumCourse) \sqsubseteq D_1$,

and the rule

$$EngStud(s) \wedge Want(s,c) \wedge Adv(c) \wedge HumCourse(c) \wedge D_1(s) \Rightarrow P_{in}. \ \square$$

The following theorem shows that our algorithm returns a terminology and an input consistency constraint that are equivalent to the original tgd. That is, for any set of inputs, if the tgd T is violated, then the predicate P_{in} will be entailed as a result of adding the terminology and rules computed by procedure **tgd-to-horn**(T).

Theorem 3: *Let $\mathcal{R}$ be a set of extended Horn rules in* CARIN*, and let T be a separable tgd. Let Δ be the set of extended Horn rules and the terminology returned by procedure* **tgd-to-horn***(T). Then, for any set of inputs G, $\mathcal{R} \cup G \models \neg T$ if and only if $\Delta \cup \mathcal{R} \cup G \models P_{in}$.* □

Proof: The proof is based on the fact that the following logical equivalence holds, where $X_1, \ldots, X_n$ are the variables that are common to ϕ and ψ, and $C_{X_1}, \ldots, C_{X_n}$ are the concepts mentioned in the procedure **tgd-to-horn**.

$$\forall X_1, \ldots, X_n \, [\phi \equiv C_{X_1}(X_1) \sqcap \ldots \sqcap C_{X_n}(X_n)] \tag{1}$$

Recall that ϕ also contains variables other than $X_1, \ldots, X_n$, which are existentially quantified. Observation 2 enables us to reformulate ϕ as a conjunction of cr-formulas. An induction on the size of the cr-formulas, shows that algorithm **tgd-to-horn** creates a concept C_{X_i} which is logically equivalent to the c_r-formula of X_i. Hence, Equation 1 holds.

For the first direction of the theorem, suppose that $\mathcal{R} \cup G \models \neg T$. That is, for every model I of $\mathcal{R} \cup G$, there exists an assignment θ_I of the variables $X_1, \ldots, X_n$, such that $I \not\models \theta_I(\phi)$ and $I \models \theta_I(\psi)$ and hence, because of Equation 1, there exists a j, $1 \leq j \leq n$ such that $I \not\models \theta_I(C_{X_j}(X_j))$. The terminology returned by procedure **tgd-to-horn** implies that $I \models \theta_I(D_{X_j}(X_j))$. Since the rule $\psi \wedge D_{X_i}(X_i) \Rightarrow P_{in}$ is in Δ, it follows that $I \models P_{in}$. Since this holds for every model I, it follows that $\Delta \cup \mathcal{R} \cup G \models P_{in}$.

For the other direction, suppose that $\Delta \cup \mathcal{R} \cup G \models P_{in}$. Assume by contradiction that $\mathcal{R} \cup G \not\models \neg T$. Therefore, there exists a model I of $\mathcal{R} \cup G$ such that for every variable assignment θ for $X_1, \ldots, X_n$, either $I \not\models \theta(\psi)$ or $I \models \theta(\phi)$. If $I \not\models \theta(\psi)$, then $I \not\models P_{in}$, because all the rules involving P_{in} have ψ in their antecedent. If $I \models \theta(\phi)$, then, by Equation 1, $I \not\models \theta(D_i(X_i))$ for every i, $1 \leq i \leq n$. In this case it also follows that $I \not\models P_{in}$, because every rule involving P_{in} has a D_i atom in its antecedent. Hence, it must be the case that $\mathcal{R} \cup G \models \neg T$. □

8 Conclusions

This paper described a new perspective on the problem of verifying Horn-rule knowledge bases, by relating it to the problem of query containment. This relationship had two major results. First, it enabled us to unify different aspects of the VVT problem, namely, I/O consistency constraints, I/O dependency constraints, and to a certain extent, input completeness constraints. Second, the relationship provided the core computational characterization of these instances of the VVT problem. In particular, we showed how the complexity of the

problem depends on the properties of the Horn rules, including the presence of interpreted predicates, negation on base predicates and recursion. Furthermore, we obtained the first sound and complete algorithm for verifying hybrid rules in a language combining Horn rules and description logics. Finally, we have also shown that by using containment in the context of hybrid knowledge bases, it is possible to obtain new decidability results concerning the problem of entailment of tuple-generating dependencies.

8.1 Related work

In this paper we considered three forms of the VVT problem. Only the VVT problem in the presence of I/O consistency constraints has received significant attention in the literature. As for the other forms of the problem, we are the first to treat input completeness constraints, and I/O dependency constraints were considered only very little. In particular, the need for verifying a knowledge base w.r.t. I/O dependency constraints has been pointed out in [24, 12]. It should be noted that testing a knowledge base w.r.t. a set of test cases can be seen as a very restricted case of the I/O dependency VVT problem, but the algorithms considered to perform such testing simply apply the KB to the (finite set of) test cases.

We now compare our work to the related work on the I/O consistency VVT problem along several axes.

The form and semantics of the rules: This paper considered only rules whose semantics is given within first-order logic.

Several works have considered the verification of OPS5-style production rules (e.g, [30, 14, 31]). In such rules, the right hand side of the rules is an *action* that may also delete facts. Ginsberg and Williamson [14] identified a subset of OPS5 rules that can be analyzed as logical rules, and presented an algorithm to do so. Their work did not consider recursive rules or interpreted predicates.

Verification of *non recursive* logical rule knowledge bases has originally been considered by Ginsberg [13] and Rousset [25]. A sound and complete algorithm for verifying non recursive Horn rules with interpreted predicates was given in [23, 39]. As stated earlier, these works did not establish the complexity of the verification problem. In particular, the complexity of the algorithms presented in [23, 39] was shown to be exponential time (by a simple reduction from the complexity of the ATMS algorithm being used). In contrast, our work provides a tight complexity bound on this problem which is Π_p^2.

Some of the subtleties involved in verifying hybrid knowledge bases have been pointed out in [17, 26]. Our work provides the first sound and complete algorithm for verifying hybrid knowledge bases.

Definition of the verification problem: On the surface, our definition of the VVT problem in the presence of I/O consistency constraints varies slightly from previous definitions (e.g., [25, 13, 14, 23]). The definition in those works did not distinguish between the predicates P_{in} and P_{out}. Instead they used a single predicate, called *bad*, to define illegal sets

of ground facts. The ground facts for which *bad* was defined could be either a set of inputs, *or* the set of facts inferred from the knowledge base (which includes the inputs). A set of rules is said to be verified if, for any set of inputs, the knowledge base does not entail *bad*. It is easy to reformulate this definition of the VVT problem into our formalism. In particular, in rules defining *bad* that contain only base predicates in their antecedents we replace *bad* by P_{in}. In the other rules, *bad* is replaced by P_{out}.

Loiseau and Rousset [23] describe a variant on the above definition. They identify a subset of the rules in the knowledge base Δ_{sure} as being *sure rules* (e.g., rules that have been previously verified). A knowledge base Δ is said to be verified if for any set of inputs G, if $G \cup \Delta_{sure} \not\models bad$, then $G \cup \Delta \not\models bad$. This definition can be reformulated in our framework as follows. We consider every rule r defining *bad*. If $Rules(r) \subseteq \Delta_{sure}$, then we replace *bad* in the consequent of r by P_{in}. Otherwise, we replace *bad* by P_{out}.

Verification algorithms: In this paper we relate the VVT problem to the problem of query containment, and therefore show that algorithms for query containment can be used for the VVT problem and vice versa. It is instructive to take a closer look at the actual algorithms used in the literature for each of these problems. In the VVT community, most of the work has used algorithms based on Assumption-based Truth Maintenance Systems (ATMS) [9]. In the database community, containment algorithms are usually explained in terms of *representative databases*. There are several points to note in a comparison:

1. The exposition of the query containment algorithms in the literature has usually been for the purpose of analyzing the complexity of the problem. However, in the cases of non recursive Horn rules, an implementation of the query containment algorithm would actually be very similar to an implementation based on ATMS. In this case, the contribution of our work is mostly the establishment of the complexity of the VVT problem.

2. In order to apply ATMS techniques for recursive rules, one has to devise a termination condition for the generation of labels (or unfoldings). In this case, the termination condition described in [8] can be used as a basis for developing an ATMS-based algorithm for VVT.

3. For hybrid knowledge bases, no extension of ATMS algorithms has been considered. In this case, the only existing algorithm is the one based on query containment [19].

8.2 Future work

There are two main directions in which our framework should be extended. As mentioned above, one direction is to explore in more detail the algorithmic aspects of the correspondence between the query containment and the VVT problem. The other direction is to find other families of constraints for which the corresponding VVT problem can be reformulated as a query containment problem. In particular, one class of constraints that is very useful in

practice (and has received little attention in the VVT literature) is output completeness constraints.

Verifying a knowledge base is only the first step in assisting its designer. When a knowledge base has been deemed as not verified, the system should aide the designer in debugging the knowledge base. An important direction that we are pursuing is to adapt the algorithms we have considered in such a way that they show the designer the flaws in the knowledge base. In particular, the algorithms we described can be modified to return a *counter example* set of inputs in cases in which the knowledge base is not verified. An interesting tradeoff in this case is whether to present the user with a counter example set of inputs, or to show her *how* the inconsistency in the knowledge base can be derived. The latter approach, for hybrid knowledge bases has been considered in [27]. Finally, the system can also propose to the designer refinements to the knowledge base that would make it consistent [5, 41].

References

[1] Alfred Aho, Yehoshua Sagiv, and Jeffrey D. Ullman. Equivalence of relational expressions. *SIAM Journal of Computing*, (8)2:218–246, 1979.

[2] Marc Ayel and Marie-Christine Rousset, editors. *Proceedings of the European Symposium on Verification and Validation of Knowledge Based Systems, EUROVAV–95*, 1995.

[3] F. Baader and B. Hollunder. A terminological knowledge representation system with complete inference algorithm. In *In Proceedings of the Workshop on Processing Declarative Knowledge, PDK-91, Lecture Notes in Artificial Intelligence*, pages 67–86. Springer-Verlag, 1991.

[4] Catriel Beeri and Moshe Vardi. A proof procedure for data dependencies. *Journal of the ACM*, 31(4):718–741, 1984.

[5] Fatma Bouali, Stephane Loiseau, and Marie-Christine Rousset. Verification and revision of rule bases. In *Proceedings of the 17th British Computer Society Conference on Expert Systems, Cambridge, United Kingdom*, 1997.

[6] Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.

[7] A.K. Chandra and P.M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[8] Surajit Chaudhuri and Moshe Vardi. On the equivalence of recursive and nonrecursive datalog programs. In *The Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego, CA.*, pages 55–66, 1992.

[9] Johan de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28, 1986.

[10] E.W Dijkstra. *A Discipline of Programming.* Prentice Hall, 1976.

[11] R. Fagin. Horn clauses and database dependecies. *Journal of the ACM*, 29(4):952–983, 1982.

[12] D. Fensel, A Schonegge, R. Groenboom, and B. Wielenga. Specification and verification of kbs. In *Proceedings of the ECAI-96 workshop on Validation, Verification and Refinement of KBS*, 1996.

[13] Allen Ginsberg. Knowledge base reduction: A new approach to checking knowledge bases for inconsistency and redundancy. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988.

[14] Allen Ginsberg and Keith Williamson. Inconsistency and redundancy checking for quasi-first-order-logic knowledge bases. *International Journal of Expert Systems: Research and Applications*, 6, 1993.

[15] Y. Gurevich and H. R. Lewis. The inference problem for template dependencies. In *Proceedings of the First ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 221–229, 1982.

[16] A. Klug. On conjunctive queries containing inequalities. *Journal of the ACM*, pages 35(1): 146–160, 1988.

[17] Sunro Lee and Robert M. O'Keefe. Subsumption anomalies in hybrid knowledge bases. *International Journal of Expert Systems: Research and Applications*, 6, 1993.

[18] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Jose, CA*, 1995.

[19] Alon Y. Levy and Marie-Christine Rousset. CARIN: a representation language integrating rules and description logics. In *Proceedings of the European Conference on Artificial Intelligence, Budapest, Hungary*, 1996.

[20] Alon Y. Levy and Marie-Christine Rousset. The limits on combining recursive horn rules and description logics. In *Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence*, 1996.

[21] Alon Y. Levy and Yehoshua Sagiv. Queries independent of updates. In *Proceedings of the 19th VLDB Conference, Dublin, Ireland*, pages 171–181, 1993.

[22] Alon Y. Levy and Dan Suciu. Deciding containment for queries with complex objects and aggregations. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Tucson, Arizona.*, 1997.

[23] Stephane Loiseau and Marie-Christine Rousset. Formal verification of knowledge bases focused on consistency: Two experiments based on ATMS techniques. *International Journal of Expert Systems: Research and Applications*, 6, 1993.

[24] Christine Pierret. Correctness of methods w.r.t problem specifications. In *Proceedings of the ECAI-96 workshop on Validation, Verification and Refinement of KBS*, 1996.

[25] Marie-Christine Rousset. On the consistency of knowledge bases: the COVADIS system. In *Proceedings of the 8th European Conference on Artificial Intelligence (ECAI 88), Munich, Germany*, 1988.

[26] Marie-Christine Rousset. Knowledge formal specifications for formal verification: a proposal based on the integration of different logical formalisms. In *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI 94), Amsterdam, Netherlands*, 1994.

[27] Marie-Christine Rousset and Pascale Hors. Modeling and verifying complex objects: A declarative approach based on description logics. In *Proceedings of the European Conference on Artificial Intelligence, Budapest, Hungary*, 1996.

[28] Y. Sagiv and M. Yannakakis. Equivalence among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655, 1981.

[29] Yehoshua Sagiv. Optimizing datalog programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 659–698. Morgan Kaufmann, Los Altos, CA, 1988.

[30] James G. Schmolze and Wayne Snyder. A tool for testing confluence of production rules. In *Proceedings of the European Symposium on Validation and Verification of KBS, EUROVAV-95*, 1995.

[31] James G. Schmolze and Wayne Snyder. Detecting redundant production rules. In *Proceedings of the AAAI Fourteenth National Conference on Artificial Intelligence*, 1997.

[32] Oded Shmueli. Equivalence of datalog queries is undecidable. *Journal of Logic Programming*, 15:231–241, 1993.

[33] Odysseas G. Tsatalos, Marvin H. Solomon, and Yannis E. Ioannidis. The GMAP: A versatile tool for physical data independence. *VLDB Journal*, 5(2):101–118, 1996.

[34] Jeffrey D. Ullman. *Principles of Database and Knowledge-base Systems, Volumes I, II.* Computer Science Press, Rockville MD, 1989.

[35] Jeffrey D. Ullman. Information integration using logical views. In *Proceedings of the International Conference on Database Theory*, 1997.

[36] Ron van der Meyden. The complexity of querying indefinite data about linearly ordered domains. In *The Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego, CA.*, pages 331–345, 1992.

[37] Ron van der Meyden. *The Complexity of Querying Indefinite Information: Defined Relations Recursion and Linear Order.* PhD thesis, Rutgers University, New Brunswick, New Jersey, 1992.

[38] Moshe Vardi. The implication and finite implication problems for typed template dependencies. *Journal of Computer and System Sciences*, 28(1):3–28, 1984.

[39] Keith Williamson and Mark Dahl. Knowledge base reduction for verifying rule bases containing equations. In *Proceedings of the AAAI-93 workshop on Validation and Verification of KBS*, 1993.

[40] M. Yannakakis and C. H. Papadimitriou. Algebric dependencies. *Journal of Computer and System Sciences*, 25(1):2–41, 1980.

[41] Neli Zlatareva. Explaining anomalies as a basis for KB refinement. In *Proceedings of the ECAI-96 workshop on Validation, Verification and Refinement of KBS*, 1996.