Verification of Knowledge Bases: a Unifying Logical View

Alon Levy and Marie-Christine Rousset

AI Principles Research Department, AT&T Laboratories
600 Mountain Avenue, Room 2C-406
Murray Hill, NJ, 07974, USA
email: levy@research.att.com
&
L.R.I - U.R.A C.N.R.S, University of Paris-Sud
Building 490
91405 Orsay Cedex
email: mcr@lri.fr

ABSTRACT : 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, they can be approached by a logical analysis of its contents. A logical analysis of the knowledge base and constraints that are known to hold on the domain enables us to detect anomalies or discrepancies between the knowledge represented in the KB and the domain. This paper describes a unified logical framework for the verification problem of knowledge bases represented by logical rules (i.e., Horn rules and some extensions). We consider several instances of the verification problem, describe algorithms for verification, and establish the computational complexity of the verification problem. In particular, we consider the verification w.r.t. *consistency* constraints that are specified separately on the inputs and on the outputs. Next, we consider *dependency* constraints on the relations between inputs and outputs. Finally, we consider several extensions to our logical formalism that enable us to express richer classes of constraints. The extensions involve the use of negation and the use of constructors of description logics in the constraints.

KEYWORDS : verification, containment, consistency and dependency constraints

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, they can be approached by a logical analysis of its contents. A logical analysis of the knowledge base and constraints that are known to hold on the domain enables us to detect anomalies or discrepancies between the knowledge represented in the KB and the domain. Several classes of such constraints can be considered. They can describe restrictions on *inputs* to the knowledge or restrictions on legal *outputs* the knowledge base produces. Alternatively, the constraints can describe *dependencies* between the inputs and outputs. The space of verification problems varies depending on the representation language used in the knowledge base and in the specification of the constraints.

This paper describes a unified logical framework for the verification problem of knowledge bases represented by logical rules (i.e., Horn rules and some extensions). We consider several instances of the verification problem, describe algorithms for verification, and establish the computational complexity of the verification problem. In particular, we consider the verification w.r.t. *consistency* constraints that are specified separately on the inputs and on the outputs. Next, we consider *dependency* constraints on the relations between inputs and outputs. Finally, we consider several extensions to our logical formalism that enable us to express richer classes of constraints. The extensions involve the use of negation and the use of constructors of description logics in the constraints.

Following our previous work [LR96b], the main tool we use to obtain our results is the connection that we establish between the verification problem and the problem of *query containment*, that has been studied in the database literature [CM77, ASU79, SY81, Klu88, Sag88, vdM92, CV92, Shm93, LS93, LS97]. We show that viewing the verification problem as a containment problem provides a uniform view of the verification problem which covers different kinds of consistency constraints on inputs and outputs as well as different dependency constraints between inputs and outputs.

This paper extends [LR96b] in several ways to provide a more comprehensive treatment of the verification problem. First, we present the general framework for the verification problem (Section 2) and show in Section 2.1 how the results of [LR96b] fit in the framework. Second, we consider dependency constraints (Section 2.2) that were not considered in our previous work. Finally, in Section 4 we discuss constraints with negation that were not discussed previously, and are important for expressing constraints on inputs and outputs.

2 The verification problem

We consider knowledge bases that are represented as 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 are safe, i.e., a variable that appears in $\bar{Y}$ appears also 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. The problem instances (i.e, the inputs of the KBS) 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. The semantics of our knowledge bases is the standard first-order logic semantics. An *interpretation* I of a knowledge base Δ contains a non-empty domain $\mathcal{O}^I$. It assigns a 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 $a \in \Delta$. 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 every atom of the antecedent of r, then $\alpha(\bar{Y}) \in q^I$, where $q(\bar{Y})$ is the consequent of r. Finally, I is a model Δ if it is a model of every rule in Δ.

An input fact base G is a set of ground atomic facts for the base predicates. Given an input fact base G, the corresponding output is a particular model of the knowledge base, called the *least fixpoint model.* This is the minimal model of the KB in which the facts of G are true. For the Horn-rule knowledge bases that we consider, this minimal model exists and is unique. It can be obtained in a constructive way by applying the rules in a forward chaining manner until no new facts can be derived. That is, we consider each rule, and whenever the antecedent can be matched with facts in the KB, we derive the consequent and add it to the KB. We refer to the model constructed in this way as the least fixpoint model.

In Section 4 we consider rules that may have negation in the antecedents. In this case, we assume the rules are stratified [Ull89]. In the presence of negation, the minimal fixpoint model is not unique, and the evaluation of the rules as described above produces a particular fixpoint model called the *perfect model.*

The verification problem for a knowledge base $\mathcal{R}$ can be defined with respect to a logical formula $\mathcal{C}$ representing constraints that hold on inputs and outputs in the domain.

Definition 1: *Given a set of Horn rules $\mathcal{R}$ representing a knowledge base, and a logical formula $\mathcal{C}$ representing constraints, the knowledge base is verified if, for any set of input facts G, $\Delta \cup G \models \mathcal{C}$.*

This definition covers different instances of the verification problem, which vary depending on the form of the constraints $\mathcal{C}$ that are considered. Clearly, when the constraint $\mathcal{C}$ may be an arbitrary first order logic formula, the verification problem is undecidable. The contribution of this paper is to investigate some classes of constraints $\mathcal{C}$ that are useful in practice and for which we show that the verification problem is decidable.

In most of the previous works about knowledge base verification, the problem was restricted to *consistency constraints* specifying legal inputs and outputs. In that setting, a knowledge base is considered to be verified if it does not enable entailing illegal outputs from legal inputs. In most works, the constraints defining legal inputs and outputs are specified using Horn rules whose consequent is a *bad* predicate. An input (or output) is considered to be legal if the bad predicate is not entailed.

In this work, in addition to consistency constraints, we also consider the verification problem w.r.t. to certain classes of *dependency constraints*, which express explicit links between legal inputs and the intended corresponding outputs. 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 given a certain input. For both of those subcases, we show that there is a close connection between the verification problem and the query containment problem. This connection leads to tight complexity results and new algorithms for the verification problem.

2.1 The verification problem w.r.t. consistency constraints

We consider that we have consistency constraints on inputs and outputs in the form of Horn rules defining two special predicates of arity 0, P_{in} and P_{out}, which are added to the knowledge base Δ. A set of input facts G is considered to be a legal input if $\Delta \cup G \not\models P_{in}$. The corresponding output $\Delta \cup G$ is said to be legal if $\Delta \cup G \not\models P_{out}$.

We can now formally define the knowledge base verification problem w.r.t. to consistency constraints:

Definition 2: *Let Δ be a knowledge base containing the predicates, P_{in} and P_{out}, describing constraints on legal inputs and outputs. The knowledge base Δ is said to be verified w.r.t. those consistency constraints if, for any set of input facts G, for which $\Delta \cup G \not\models P_{in}$, then $\Delta \cup G \not\models P_{out}$.*

Note that Definition 2 corresponds to the case where $\mathcal{C}$ is the formula $P_{out} \Rightarrow P_{in}$. Our definition of the verification problem w.r.t consistency constraints differs slightly from previous definitions that were proposed in the literature (e.g., [Gin88, Rou88, GW93, LR93]). 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. However, it can be shown that the existing definitions of consistency-based verification using a single predicate for defining illegal inputs and outputs can be easily reformulated in our setting. As we see shortly, another advantage of our formulation is that it makes the connection 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 $HumCourse$. The input atom $Want(s,c)$ denotes that student s wants to take course c during the current year, and $Prev(s,c)$ is an input atom which denotes that s has already taken c in a previous year. The atom $Take(s,c)$ denotes that s will take course c. Suppose that the following rules describe our domain.

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

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 exhibit the following input, which is legal (w.r.t to the input constraints that we consider), for which we can derive an illegal output:
$\{Want(S_1,C_2), HumStud(S_1), Adv(C_2), Prev(S_1,C_1), PrereqOf(C_2,C_1), EngCourse(C_2)\}$
This input represents a case of a student S_1 who 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 illegal.

2.2 The verification problem w.r.t. dependency constraints

Verifying that a knowledge base satisfies consistency constraints is useful but it is not enough to guarantee that the knowledge base is correct. In particular, the consistency constraints are too loose to express dependencies which are known to exist between legal inputs and their corresponding outputs.

Example 2: Continuing with our example, suppose we have a constraint stating that students who are in their first two years and who have previously taken one advanced course must necessarily take atleast one basic course. Formally, we could state that constraint with the following logical formula:

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

A different way to formulate the constraint is to define two special predicates In and Out, respectively defining the left hand side and the right hand side of the implication. We obtain the following formulation:

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

Our dependency constraint can now be expressed as the implication $\forall s(In(s) \Rightarrow Out(s))$.

In the KBS verification community, verifying knowledge bases w.r.t. dependency constraints between inputs and outputs has received very little attention. In some sense, testing a knowledge base w.r.t. a set of test cases can be seen as a very restricted case of verifying a knowledge base w.r.t. very particular ground dependency constraints. On the other hand, for verification of conventional programs, it is usual to state preconditions and postconditions on programs, and to check that whenever the preconditions are satisfied on the inputs, the postconditions are satisfied on the outputs. Our approach for defining the knowledge base verification problem w.r.t. dependency constraints is similar in spirit.

We consider that we have dependency constraints between legal inputs and outputs in the form of Horn rules defining pairs of special predicates $\{In_i,\ Out_i\}$, which are added to the knowledge base Δ. For a given i, the predicates In_i and Out_i have the same arity. The union of the antecedents of the rules having $In_i(\bar{X}_i)$ as consequent expresses a certain precondition on inputs. The union of the antecedents of the rules having $Out_i(\bar{X}_i)$ as consequent expresses the postcondition that the outputs corresponding to the inputs satisfying that precondition have to satisfy. The elements of the tuple $\bar{X}_i$ denotes the elements on which the dependency holds. In our previous example, they correspond to the universally quantified variable s in the initial logical formula, which denotes a student.

The dependency constraints that we consider do not cover all the possible dependency constraints that could be expressed in first order logic. They allow us to express a conjunction of constraints of the form

$$\begin{aligned}&\forall \bar{X}[\exists \bar{Y}_1(p_{11}(\bar{Y}_{11}) \wedge \ldots \wedge p_{1j_1}(\bar{Y}_{1j_1})) \vee \ldots \vee \exists \bar{Y}_k(p_{k1}(\bar{Y}_{k1}) \wedge \ldots \wedge p_{kj_k}(\bar{Y}_{kj_k}))\\ &\quad \Rightarrow \exists \bar{Z}_1(q_{11}(\bar{Z}_{11}) \wedge \ldots \wedge q_{1j_1}(\bar{Z}_{1j_1})) \vee \ldots \vee \exists \bar{Y}_l(q_{l1}(\bar{Z}_{l1}) \wedge \ldots \wedge q_{lj_l}(\bar{Z}_{lj_l}))]\end{aligned}$$

where for every tuple $\bar{Y}_m$ and $\bar{Z}_n$, $\bar{Y}_m \subset \bar{Y}_{m1} \cup \ldots \cup \bar{Y}_{mj_m}$ and $\bar{Z}_n \subset \bar{Y}_{n1} \cup \ldots \cup \bar{Y}_{nj_n}$, and the other variables appearing in the tuples $\bar{Y}_{mj}$ and $\bar{Z}_{nk}$ are variables of the tuple $\bar{X}$.

We can now formally define the knowledge base verification problem w.r.t. dependency constraints:

Definition 3: *Let Δ be a knowledge base containing rules defining pairs of predicates $(In_1, Out_1), \ldots, (In_k, Out_k)$ describing dependencies between legal inputs and outputs. The knowledge base Δ is said to be verified w.r.t those dependency constraints if, for each $i \in [1 \ldots k]$, for any set of input facts G, if $\Delta \cup G \models In_i(\bar{a})$, then $\Delta \cup G \models Out_i(\bar{a})$.*

Note that Definition 3 corresponds to the instance of Definition 1 where the formula $\mathcal{C}$ is

$$\bigwedge_{i=1}^{k} (\forall \bar{X})[In_i(\bar{X}) \Rightarrow Out_i(\bar{X})].$$

In the same way, we can express dependency constraints between legal outputs and the corresponding legal inputs. We then obtain a similar verification problem, in which the roles of the predicates In and Out are reversed.

3 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. The query containment problem is to decide whether in any minimal fixpoint model of the knowledge base, the extension of one predicate contains the extension of another. It has been extensively considered in database theory because it is a core technique for query optimization and related problems. Formally, given a knowledge base Δ 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 4: *Let P_1 and P_2 be two predicates of the same arity in the knowledge base Δ. 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 definition says that whenever P_1 is entailed by $\Delta \cup G$, then so is P_2.

The following theorems formalize the connection between the verification problem (w.r.t. both consistency and dependency constraints) and the containment problem.

Theorem 1: *Let Δ be a knowledge base with predicates P_{in} and P_{out} describing legal inputs and outputs. The knowledge base Δ is verified w.r.t those consistency constraints if and only if $P_{out} \subseteq P_{in}$.*

Theorem 2: *Let Δ be a knowledge base with pairs of predicates $(In_1, Out_1), \ldots, (In_k, Out_k)$ describing dependencies between legal inputs and outputs. The knowledge base Δ is verified w.r.t those dependency constraints if and only if for every $i \in [1 \ldots k], In_i \subseteq Out_i$.*

The connection between the verification and containment problems yields several core decidability and complexity results, which have been shown in [LR96b]. They are summarized in the following table. It is important to emphasize that the complexity results consider the size of the *rules* and not the size of the ground facts. In the table, $Rules(P)$ denotes the set of rules relevant to P in $\mathcal{R}$.[1]

Rules(P_{out}) / Rules(P_{in})	no recursion + no interpreted predicates	interpreted predicates	recursion	recursion + interpreted predicates
no recursion + no interpreted predicates	NP-complete	NP-complete	doubly exponential	undecidable
interpreted predicates	$\Pi_2{}^P$	$\Pi_2{}^P$	undecidable	undecidable
recursion	NP-comp	$\Pi_2{}^P$	undecidable	undecidable
recursion + interpreted predicates	doubly exponential	doubly exponential	undecidable	undecidable

[1] $Rules(P)$ is the minimal subset of $\mathcal{R}$ that satisfy the following conditions: (a) If P is the predicate in the consequent of the rule r, then $r \in Rules(P)$, (b) If the predicate Q appears in a rule $r \in Rules(P)$, then any rule whose consequent has Q is also in $Rules(P)$.

4 Specifying richer input and output constraints

An important aspect of the knowledge base verification problem is how input and output constraints are described. In our definition so far, and in most previous work in the field, the constraints were specified by Horn rules defining special predicates P_{in} and P_{out} to describe consistency constraints, or In_i and Out_i to describe dependency constraints. However, Horn rules are not always expressive enough for describing constraints that arise in applications.

In this section, we investigate two ways to express and deal with more complex constraints than those previously considered. In these extensions we preserve the connection with the containment problem (albeit in an extended language), and we are therefore able to extend out decidability and complexity results.

A first way is to extend our language with negation. In particular, it enables dealing with richer consistency constraints describing legal inputs and outputs than those previously considered. The existing results about containment in presence of negation allow us to identify cases where this extension of the verification problem is decidable and cases where it is not.

A second way is to extend our Horn rule language with description logics. We have shown in [LR96b] that having CARIN extended rules [LR96a], in which some predicates appearing in the antecedents of the rules can be described as concepts or roles in a description logics terminology, enables us to express both more complex consistency and dependency constraints.

4.1 Input and Output constraints with negation

The following example illustrates the need for negation to express richer input and output constraints. It illustrate the use of negation to express *completeness* constraints on the inputs, i.e., that the presence of certain inputs must imply that others are present as well. The example also illustrates how negation is used to express some dependency constraints as consistency constraints.

Example 3: Suppose we want to express the constraint stating that all the courses that have been previously taken by students who are in their first year are mandatory for that year. It can be expressed by the following Horn rule relating some inputs facts to other input facts:

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

A possible way to express it as a Horn rule with the special predicate P_{in} as consequent is to allow negation:

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

In the same way, we can express similar constraints on outputs (i.e., to relate the presence of some output facts to the presence of other output facts by a Horn rule). A similar solution is possible to express them by Horn rules with negation having the predicate P_{out} as consequent.

Consider the use of negation for expressing dependency constraints. Suppose we want to express the dependency constraint stating that if the KB outputs that a student will take a course which is optional, then the input must state that the student *wants* to take the course. This can be expressed by the following formula:

$$Take(s,c) \wedge Optional(c) \Rightarrow Want(s,c).$$

Allowing for negation enables translating that dependency constraint into the following consistency constraint:

$$Take(s,c) \wedge Optional(c) \wedge \neg Want(s,c) \Rightarrow P_{out}.$$

The connection between the verification problem and the query-containment problem still holds in the presence of negation. Unfortunately, when negation is present, the containment problem becomes undecidable when negation is applied to non-base predicates. It is important to note that in the case of the input constraints, it is enough to allow negation on base predicates only. The following theorem points out the main decidable case for verification in the presence of negation.

Theorem 3: *Let Δ be a knowledge base with safe negation applied only to base predicates*[2]. *Assume Δ defines the pairs of predicates $(In_1, Out_1), \ldots, (In_k, Out_k)$ describing dependencies between legal inputs and outputs. If the rules in Δ are not recursive, then the verification problem is decidable and is Π_2^P complete.*

[2] A rule is safe if every variable that appears in a negated subgoal appears also in a positive subgoal.

The proof of Theorem 3 follows from the containment algorithm described in [LS93]. Before ending this section, two points should be noted. First, the algorithm presented in [LS93] would remain *sound* also for rules in which negation is applied to non-base predicates. That is, if the algorithm returns that the knowledge base is verified, then indeed it is. Furthermore, this algorithm is complete when we consider the condition of *uniform containment* [Sag88]. Uniform containment considers the subset relationship in *all* models of the Δ, not only the minimal fixpoint model. The second point to note is that we can also apply negation to non-base predicates that are defined by rules without *projection*. A rule does not perform a projection if each variable that appears in the antecedent of the rule also appears in its consequent. It follows from [SY81] that the verification problem will remain decidable for non-recursive knowledge bases when negation is applied to predicates defined only with rules with no projection.

4.2 CARIN extended rules for input and output constraints

The following example illustrates a *tuple generating constraint,* (tgd) which is also a common type of constraint encountered in practice.

Example 4: 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 formula:

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

The formula is called a tuple generating dependency because existentially quantified variables appear in the right-hand side of the rule.

In [LR96b] we define a class of tgd's for which we can translate the verification problem to a containment problem on extended Horn rules in CARIN [LR96a]. CARIN rules are Horn rules in which some predicates appearing in the antecedents are concepts or roles defined in a description logic.

Informally, description logics enable us to define the class of elements that *do not* satisfy the right hand side of the tgd. In a description logic we can define *complex classes*. For instance, 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 fillers of the property $Prev$ do not belong to the class $Basic \sqcap HumCourse$. We use the class C_{tgd} as a predicate in an extended Horn rule. Specifically, in our example, the tgd can be translated into the following extended rule:

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

In the same paper [LR96b], we give a complete verification algorithm for hybrid knowledge bases composed of CARIN extended rules.

It is important to note that the advantage of CARIN goes beyond expressing input constraints as considered in [LR96b]. It also makes it possible to express more complex dependency constraints than those we considered so far, as it is illustrated in the following example.

Example 5: Suppose we want to express the 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 formulas:

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

The first formula is not a Horn rule. However, it can be expressed by the following extended CARIN Horn rule if we define the concept C by the description $\forall Prev.Basic$:

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

It is important to note that description logics enable us to express certain kinds of negation. However, an interesting aspect of description logics is that the limited way in which they introduce negation still enables to decide containment (i.e., subsumption), even when combined with Horn rules. In contrast, as we have seen in the previous section, adding negation to antecedents of Horn rules (even when it is stratified), makes the containment problem undecidable. A full comparison between these forms of negation is beyond the scope of this paper.

5 Conclusions

We described a logical framework that unifies different variants of the knowledge base verification problem. In particular, we considered consistency constraints as well as dependency constraints, thus covering different aspects of knowledge base correctness and completeness. We gained insight into the verification problem by showing that it is closely related to the problem of query containment. Finally, we have shown that with hybrid knowledge bases combining rules and description logics, we are able to handle richer constraints than those that have been considered in previous work on verification.

It is important to note that many of the issues related to verifying conceptual models expressed in formal languages turned out to coincide with representational issues for verification. This is not surprising since we are faced with the same problem: verification is reduced to some formal analysis of the language used to express the knowledge. In particular, the need for verifying the control knowledge w.r.t dependency constraints has been pointed out in [Pie96, FSGW96].

There are different directions for future work. First, a thorough comparison between ATMS-based algorithms and containment algorithms has the potential to improve both existing consistency checking algorithms and existing containment algorithms. In the KBS verification community, the use of ATMS-based techniques has been extensively considered for verifying the consistency of rule bases ([Rou88, Gin88, Zla92, Mes92, AV93]). The basic idea underlying an ATMS (Assumption-based Truth Maintenance System) [dK86] is, given a set of rules, to calculate (and update) the so-called *label* of each deducible literal. From a logical point of view, the label of L is a minimal consistent disjunctive normal form, which represents necessary and sufficient conditions for the rule base to derive the literal L from base literals. An ATMS can be seen as an incremental machinery for unfolding the rules allowing one to obtain, from a given set of rules, an equivalent set of rules in which all the antecedents of rules contain only base predicates. The particularity of ATMS, which explains its use for verification purposes, is that the incompatibility between literals is considered explicitly and plays a central role at each unfolding step. In particular, each antecedent of the rules which are in the process to be unfolded is checked to be consistent. The ATMS-based algorithms are well adapted for verifying consistency of large and deep collections of non recursive rules. Those techniques could be exported to obtain efficient containment algorithms in presence of rules whose depth makes a complete unfolding inefficient[3]. Conversely, the techniques used by existing containment algorithms to deal with recursion and negation could be exported to make ATMS able to consider recursive rules with variables and negation. In addition, containment algorithms can be used to perform rule base verification w.r.t. dependency constraints.

Second, the set of constraints that can be considered as the basis for verification can be extended. We are investigating more complex consistency or dependency constraints than those we considered so far. We think that other kinds of constraints can be interesting in practice, while keeping the verification problem related to the containment problem, so that the approach presented in this paper can be applied to obtain complexity results and verification algorithms in a more extended framework. In particular, an important aspect of knowledge base verification, which has received very little attention so far, is the completeness of knowledge bases. Some aspects of the completeness can be captured by more general consistency or dependency constraints on inputs and outputs.

Finally, 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, [SS95, GW93]). In such rules, the right hand side of the rules is an *action* that may delete facts from the knowledge base. Ginsberg and Williamson [GW93] identify a subset of OPS5 rules that can be analyzed as logical rules, and present an algorithm to do so. Their work does not consider recursive rules or interpreted constraints. We are interested in investigating how to integrate OPS-like rules into our framework, and in particular, to relate the verification problem of OPS-like rules with the problem of containment of active rules in database theory.

References

[ASU79] Alfred Aho, Yehoshua Sagiv, and Jeffrey D. Ullman. Equivalence of relational expressions. *SIAM Journal of Computing*, (8)2:218–246, 1979.

[AV93] Marc Ayel and Laurence Vignollet. Sycojet and sacco, two tools for verifying expert systems. *International Journal of Expert Systems: Research and Applications*, 6, 1993.

[3] Some containment algorithms are based on a first step which completely unfolds the rules before checking for containment

[CM77] 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.

[CV92] 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.

[dK86] Johann de Kleer. An assumption-based tms. *AIJ*, 28, 1986.

[FSGW96] 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.

[Gin88] Allen Ginsberg. Knowledge base reduction: A new approach to checking knowledge bases for inconsistency and redundancy. In *Proceedings of AAAI-88*, 1988.

[GW93] 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.

[Klu88] A. Klug. On conjunctive queries containing inequalities. *Journal of the ACM*, pages 35(1): 146–160, 1988.

[LR93] 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.

[LR96a] 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.

[LR96b] Alon Y. Levy and Marie-Christine Rousset. Verification of knowledge bases using containment checking. In *Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence*, 1996.

[LS93] Alon Y. Levy and Yehoshua Sagiv. Queries independent of updates. In *Proceedings of the 19th VLDB Conference, Dublin, Ireland*, pages 171–181, 1993.

[LS97] 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, to appear.*, 1997.

[Mes92] Pedro Meseguer. Incremental verification of rule-based expert systems. In *Proceedings ECAI-92*, 1992.

[Pie96] 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.

[Rou88] 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.

[Sag88] 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.

[Shm93] Oded Shmueli. Equivalence of datalog queries is undecidable. *Journal of Logic Programming*, 15:231–241, 1993.

[SS95] 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.

[SY81] Y. Sagiv and M. Yannakakis. Equivalence among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655, 1981.

[Ull89] Jeffrey D. Ullman. *Principles of Database and Knowledge-base Systems, Volumes I, II.* Computer Science Press, Rockville MD, 1989.

[vdM92] 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.

[Zla92] Neli Zlatareva. Truth maintenance systems and their application for verifying expert systems. *Artificial Intelligence Review*, 6, 1992.