# CARIN: A Representation Language Combining Horn rules and Description Logics

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**

We describe CARIN, a novel family of representation languages, that combine the expressive power of Horn rules and of description logics. We address the issue of providing sound and complete inference procedures for such languages. We identify *existential entailment* as a core problem in reasoning in CARIN, and describe an existential entailment algorithm for the $\mathcal{ALCNR}$ description logic. As a result, we obtain a sound and complete algorithm for reasoning in non recursive CARIN-$\mathcal{ALCNR}$ knowledge bases, and an algorithm for rule subsumption over $\mathcal{ALCNR}$. We show that in general, the reasoning problem for recursive CARIN-$\mathcal{ALCNR}$ knowledge bases is undecidable, and identify the constructors of $\mathcal{ALCNR}$ causing the undecidability. We show two ways in which CARIN-$\mathcal{ALCNR}$ knowledge bases can be restricted while obtaining sound and complete reasoning.

**Keywords:** description logics, Horn rules, query containment, hybrid languages, knowledge representation.

# 1 Introduction

Horn rule languages have formed the basis for many Artificial Intelligence applications as well as the basis for deductive and active database models. 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. One of the significant limitations of Horn rules is that they are not expressive enough to model domains with a rich hierarchical structure. In contrast, description logics are a family of representation languages that have been designed especially to model rich

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

hierarchies of classes of objects. The computational and expressive properties of description logics have been extensively studied. Several systems have been built based on description logics (e.g., CLASSIC [8], LOOM [28], BACK [31], KRIS [5]), and they have been used in several applications (e.g., [38, 37]).

A description logic is a subset of first order logic with equality. In a description logic we define *sets* of objects (referred to as *concepts*) as unary relations, and we define relationships between objects as binary relations (called *roles*). Concepts are defined by the necessary and sufficient conditions satisfied by objects in the set. These conditions are expressed by *concept descriptions* that are built from a set of *constructors*. For example, the description $\mathsf{Parent} \sqcap (\forall\ \mathsf{child.Smart})$ describes the instances of the concept Parent, all of whose children are instances of the class Smart. Similarly, some description logics allow *role descriptions* in addition to concept descriptions. Description logic knowledge bases contain a *terminology* in which we define the set of concepts and roles used in the knowledge base. Description logics and their properties vary depending on the set of allowed constructors and the kinds of statements allowed in the knowledge base. Much of the research in description logics has concentrated on algorithms for determining subsumption relations between concepts, satisfiability of knowledge bases and for checking membership of an object in a concept.

Horn rules and descriptions logics are two orthogonal subsets of first order logic [7]. Several applications, such as combining information from multiple heterogeneous sources, modeling complex physical devices, significantly benefit from combining the expressive power of both formalisms. Starting from the KRYPTON language [9], several works have considered the design of hybrid representation languages that combine rules with description logics (e.g., [17, 14, 2, 29]).

This paper describes CARIN, a novel family of languages, that extend Horn rules with the expressive power of description logics. CARIN knowledge bases contain *both* a set of Horn rules and a description-logic terminology. CARIN combines the two formalisms by allowing the concepts and roles, defined in the terminology, to appear as predicates in the antecedents of the Horn rules. CARIN is distinguished from previous hybrid languages in that we allow both the roles and concepts to appear in the Horn rules. In contrast, previous languages (e.g., AL-Log [14]) only allow the usage of concepts as typing constraints on variables that *already appear* elsewhere in the rules.

The semantics of CARIN is derived naturally from the semantics of its component languages. The key problem in developing such a hybrid language is designing a *sound* and *complete* inference procedure for answering queries in the language. As already observed for the simpler language AL-Log [14], combining standard Horn rule inference procedures with intermediate terminological reasoning steps does not result in a complete inference procedure for AL-Log, and therefore the same applies to CARIN knowledge bases. The reason for the incompleteness is that Horn rule reasoning procedures apply each rule in *isolation,* and they try to *instantiate* the antecedent of the rule in order to derive its consequent.

In this paper we address the reasoning problem in CARIN. We consider CARIN for the case in which the description logic is $\mathcal{ALCNR}$, (referred to as CARIN-$\mathcal{ALCNR}$). $\mathcal{ALCNR}$ is the most expressive decidable implemented description logic [5]. For the Horn-rule component, we consider function-free Horn rules. Our results distinguish the case of nonrecursive Horn rules from the case of recursive Horn rules in CARIN knowledge bases.

In the nonrecursive case, we show that a single step of applying a Horn rule needs to be replaced by a more sophisticated reasoning step. We isolate this step and refer to it as *existential entailment*. Speaking informally, the existential entailment problem is to decide whether the antecedent of a Horn rule, together with a description logic terminology logically entail the disjunction of the antecedents of a set of Horn rules. We describe an existential entailment algorithm for $\mathcal{ALCNR}$. The existential entailment algorithm entails several important results:

- It provides a sound and complete inference procedure for nonrecursive CARIN knowledge bases in which the description logic component is $\mathcal{ALCNR}$.
- A particular and important case of the above is that we provide the first algorithm for answering *arbitrary* conjunctive queries over an $\mathcal{ALCNR}$ knowledge base.
- It provides an algorithm for rule subsumption over $\mathcal{ALCNR}$, which is an important building block in several query optimization algorithms.
- The existential entailment algorithm can be combined with the *constrained-resolution* algorithm described by Bürckert [12] to yield a refutation complete SLD-resolution algorithm for recursive CARIN-$\mathcal{ALCNR}$.[1]

In the recursive case, we show how the decidability of the reasoning problem depends on the constructors allowed and on the form of the Horn rules. In particular, we show the following:

- Reasoning in recursive CARIN-$\mathcal{ALCNR}$ is undecidable. We show that in each of the following cases by itself, the reasoning problem is undecidable:
  - the description logic contains the constructor $(\forall R.C)$,
  - the description logic contains the constructor $(\leq\ n\, R)$, or
  - the terminology contains terminological cycles and either the constructor $(\exists R.C)$ or $(\geq\ n\, R)$.

  These results are significant because the constructors leading to undecidability are at the *core* of most description logics.
- We identify two ways of restricting the sentences in the knowledge base for which we establish sound and complete inference procedures:
  - The first restriction concerns the description logic. We show that if the description logic does not contain the constructors $(\forall R.C)$ and $(\leq\ n\, R)$, and the terminology contains only concept definitions that are acyclic, the reasoning problem is decidable.

[1] Recall that a refutation complete algorithm will terminate if the query is entailed by the knowledge base, but may *not* terminate otherwise.

– The second restriction concerns the form of the Horn rules. We show that if the rules are *role-safe*, then recursive Horn rules can be combined with all of $\mathcal{ALCNR}$, and reasoning is still decidable. Role-safe rules require that in every role atom at least one variable that appears in the atom also appears in an atom of a base predicate (i.e., a predicate that does not appear in the consequent of a Horn rule, and is not a concept or role predicate). It should be noted that this restriction covers some of the common usages of recursive rules (e.g., expressing graph connectivity).

CARIN has several advantages that are important in applications. The added expressive power of CARIN w.r.t. its component languages enables us to express rich constraints on classes of objects in a form of a concept hierarchy while still having the ability to use predicates of arbitrary arity, and the ability to express arbitrary joins between relations. In addition, CARIN provides a *query language* over description logic knowledge bases in which more traditional queries can be expressed. In particular, it is possible to express conjunctive queries, unions of conjunctive queries and recursive queries. In contrast, queries over description logic knowledge bases have been limited to membership and subsumption queries. Both of these features in CARIN were key to the design of the Information Manifold system [22] that combines information from multiple heterogeneous sources. Furthermore, conjunctive queries and unions thereof are the basic building block of most database query languages (e.g., SQL). Therefore, the ability to answer conjunctive queries over description logics is important when we consider systems that combine description logics with traditional databases. Finally, the ability to answer subsumption queries between conjunctive queries in CARIN has been important in developing novel algorithms for knowledge base verification [25]. We elaborate on the applications of CARIN in Section 8.

The paper is organized as follows. Section 2 presents the CARIN family of languages, and illustrates the novel kinds of inferences that can be made in the language. Section 3 describes an algorithm for existential entailment that is at the heart of several reasoning algorithms for CARIN. Section 4 considers inference in non recursive CARIN knowledge bases, and shows how existential entailment is used for a sound and complete inference procedure for non recursive CARIN-$\mathcal{ALCNR}$. Section 5 describes several cases in which allowing recursive Horn rules in CARIN knowledge bases leads immediately to undecidability of the reasoning problem. Sections 6 and 7 describe cases in which there exist sound and complete algorithms for reasoning in CARIN in the presence of recursive Horn rules. Section 8 concludes with related work and discussion.

## 2 The CARIN Languages

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 CARIN-$\mathcal{L}$ knowledge base (KB) contains three components, the first is a description-logic terminology, the second is a set of Horn rules and the third is a set of ground facts. The terminology is a set of statements in $\mathcal{L}$ about concepts and roles in the domain. Concepts and roles from the terminology can also appear as predicates in the antecedents of the Horn rules and in the ground facts. Predicates that do not appear in the terminology are called *ordinary predicates*. Ordinary predicates can be of any arity. We describe each component below.

## 2.1 Terminological Component in CARIN

The terminological component of a CARIN-$\mathcal{L}$ knowledge base contains a set of formulas in the description logic $\mathcal{L}$. 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 are built from the set of *constructors* of $\mathcal{L}$. Here we consider CARIN languages in which the description logic component is any subset of the language $\mathcal{ALCNR}$ [11]. 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 the terminological component $\Delta_{\mathcal{T}}$ of a CARIN knowledge base $\Delta$ are either *concept definitions, concept inclusions*[2] or *role definitions*. A concept definition is a statement of the form $A := D$, where $A$ is a concept name and $D$ is a concept description. We assume that a concept name appears on the left hand side of at most one concept definition. An inclusion statement is of the form $C \sqsubseteq D$, where $C$ and $D$ are concept descriptions. Intuitively, a concept definition associates a definition with a name of a concept. 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. In $\mathcal{ALCNR}$ role descriptions are limited to conjunctions of atomic roles.

A concept name $A$ is said to *depend* on a concept name $B$ if $B$ appears in the concept definition of $A$. A set of concept definitions is said to be *cyclic* if there is a cycle in the dependency relation. When the terminology contains only concept definitions and has no cycles we can *unfold* the terminology by iteratively substituting every concept name with its definition. As a result, we obtain a set of concept definitions, where all the concepts that appear in the right hand sides do not appear on the left hand side of any definition.

[2]A concept definition can also be given by two inclusion statements. However, we single out concept definitions here because they will be of special interest later on.

The semantics of the terminological component 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 $\Delta_{\mathcal{T}}$, and a binary relation $R^I$ over $\mathcal{O}^I \times \mathcal{O}^I$ to every role $R$ in $\Delta_{\mathcal{T}}$. 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 $\Delta_{\mathcal{T}}$ if $C^I \subseteq D^I$ for every inclusion $C \sqsubseteq D$ in the terminology, $A^I = D^I$ for every concept definition $A := D$, and $P^I = R^I$ for every role definition $P := R$. We say that $C$ *is subsumed by* $D$ w.r.t. $\Delta_{\mathcal{T}}$ if $C^I \subseteq D^I$ in every model $I$ of $\Delta_{\mathcal{T}}$.

**Example 2.1:** Consider the following terminology, $\mathcal{T}_1$:

european $\sqcap$ american $\sqsubseteq \bot$
european-associate := $\exists$associate.european
american-associate := $\exists$associate.american
no-fellow-company := $\forall$associate.$\neg$ american
international-company := european-associate $\sqcup$ american-associate

The concepts european and american are primitive concepts, and the first inclusion states that they are disjoint. The concept european-associate (respectively, american-associate) is defined to be the set of individuals that have at least one filler of the role associate, which is a member of european (respectively, american). The concept no-fellow-company is defined to be the set of individuals that have as fillers of the role associate only individuals which are not member of american. The concept international-company represents the set of individuals that belong either to european-associate or to american-associate. As an example of a subsumption relationship that can be inferred from the terminology, the concept european-associate $\sqcap$ american-associate is subsumed by ($\geq$ 2 associate). The subsumption holds because instances of the first concept are required to be instance of both european-associate and of american-associate, and therefore to have both an American associate and a European associate. However, since European and American companies are disjoint sets, it entails that instances of european-associate $\sqcap$ american-associate have at least two associates. $\square$

## 2.2 Horn Rules and Ground Facts in CARIN

**Horn rule component:** The Horn-rule component $\Delta_{\mathcal{R}}$ of a CARIN knowledge base $\Delta$ contains a set of Horn rules that are 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 $\bar{Y}$ must also appear in $\bar{X}_1 \cup \ldots \cup \bar{X}_n$. The predicates $p_1, \ldots, p_n$ may be either concept or role names, or *ordinary* predicates that do not appear in $\Delta_{\mathcal{T}}$. Recall that ordinary predicates can be of any arity. The predicate $q$ must be an ordinary predicate. It should be noted that CARIN Horn rules are more general than previous languages such as AL-log [14], where, in addition to ordinary predicates, only concept predicates were allowed in the rules, and a variable appearing in a concept atom had to appear in an atom of an ordinary predicate in the antecedent.

An ordinary predicate $p$ is said to *depend* on an ordinary predicate $q$ if $q$ appears in the antecedent of a Horn rule whose consequent is $p$. A set of rules are said to be *recursive* if there is a cycle in the dependency relation among ordinary predicates.

**Ground fact component:** The ground-fact component of a CARIN knowledge base contains a set of ground atomic facts of the form $p(\bar{a})$ where $\bar{a}$ is a tuple of constants and $p$ is either a concept, role or ordinary predicate.

**Example 2.2:** As an example of a CARIN-$\mathcal{ALCNR}$ knowledge base, we can consider the terminology $\mathcal{T}_1$, with the following rules, $\mathcal{R}_1$, using the ordinary predicates made-by, monopoly and price:

$r_1$ : made-by(X,Y) $\wedge$ no-fellow-company(Y) $\Rightarrow$ price(X,usa,high)
$r_2$ : made-by(X,Y) $\wedge$ associate(Y,Z) $\wedge$ american(Z) $\wedge$ monopoly(Y,X,usa) $\Rightarrow$ price(X,usa,high).

and the following ground facts:

$\mathcal{A}_1$ : {made-by(a,b), monopoly(b,a,usa), american-associate(b)} $\Box$

## 2.3 Semantics of CARIN

The semantics of CARIN is derived in a natural way from the semantics of its components languages. An interpretation $I$ contains a non-empty domain $\mathcal{O}^I$. The interpretation assigns to every constant $a$ in $\Delta$ an object $a^I \in \mathcal{O}^I$, and a relation of arity $n$ over the domain $\mathcal{O}^I$ to every predicate of arity $n$ in $\Delta$. An interpretation $I$ is a model of a knowledge base $\Delta$ if it is a model of each of it components. Models of the terminological component were defined in Section 2.1. An interpretation $I$ is a model of a rule $r$ if, whenever $\alpha$ 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 of a ground fact $p(\bar{a})$ if $\bar{a}^I \in p^I$. We make the *unique names assumption*, i.e., if $a$ and $b$ are constants in $\Delta$, then $a^I \neq b^I$.

**Remark 2.1:** It should be noted that CARIN does not allow concept and role atoms to appear in the consequents of the rules because of the underlying assumption that the terminological component *completely* describes the hierarchical structure in the domain, and therefore, the rules should not allow to make new inferences about that structure. $\Box$

## 2.4 Reasoning in CARIN

The reasoning problem we address in CARIN is the following:

> Given a CARIN knowledge base $\Delta$ and a ground atomic query of the form $p(\bar{a})$, where $p$ can be any predicate, and $\bar{a}$ is a tuple of constants, does $\Delta \models p(\bar{a})$?

The following example shows some of the additional inferences that can be drawn from CARIN rules but not from either of its sublanguages alone.

**Example 2.3:** Suppose we have the CARIN knowledge base $\mathcal{T}_1 \cup \mathcal{A}_1 \cup \mathcal{R}_1$. The following entailment holds:

$$\mathcal{T}_1 \cup \mathcal{A}_1 \cup \mathcal{R}_1 \models \mathsf{price}(\mathsf{a}, \mathsf{usa}, \mathsf{high}).$$

The fact american-associate(b) and $\mathcal{T}_1$ entail that b has some associate that is an american. Therefore, even though no rule of $\mathcal{R}_1$ can be totally *instantiated* on $\mathcal{A}_1$, the missing conjuncts of $r_2$ are *entailed* by $\mathcal{A}_1 \cup \mathcal{T}_1$.

As another example, suppose we have the ground facts:

$\mathcal{A}_2$ : {made-by(a,b),monopoly(b,a,usa), international-company(b)}.

The following entailment holds:

$$\mathcal{T}_1 \cup \mathcal{A}_2 \cup \mathcal{R}_1 \models \mathsf{price}(\mathsf{a}, \mathsf{usa}, \mathsf{high}).$$

Here, $\mathcal{T}_1 \cup \mathcal{A}_2$ does not entail the antecedent of any *single* rule in $\mathcal{R}_1$. However, we can make the inference by reasoning by cases: (1) if b has at least one American associate, then price(a,usa,high) will follow because the antecedent of rule $r_2$ will be entailed, as explained above. (2) if b has *no* American associate, then no-fellow-company(b) will be entailed, and price(a,usa,high) will follow from $r_1$. □

The above examples illustrated that there are two aspects of traditional Horn rule inference mechanisms that make them inadequate for CARIN knowledge bases. The first aspect is that they consider each rule in *isolation*, and the second aspect is that for each rule they try to *instantiate* the antecedent in order to derive the consequent. The example with $\mathcal{A}_1$ showed that a KB may entail the antecedent of a rule without the antecedent being instantiated in the KB. The example with $\mathcal{A}_2$ showed that a KB may entail the *disjunction* of the antecedents of two rules without entailing either of them. These problems have also been observed in [14], even when role predicates were not allowed in the rules. Therefore, to enable complete reasoning in CARIN, the process of instantiating a single rule needs to be replaced by an algorithm that decides whether a set of ground facts and a terminology *entails* the *disjunction* of the antecedents of a set of Horn rules. In the next section we formalize this decision problem as the *existential entailment* problem, and describe an algorithm for solving it. In Section 4 we show that the existential entailment algorithm stands at the core of several reasoning problems in CARIN.

# 3 Existential Entailment Algorithm for $\mathcal{ALCNR}$

Formally, the existential entailment problem for a description logic $\mathcal{L}$ is the following.

**Definition 3.1:** *(existential entailment) Let $\mathcal{T}$ be a terminology in the description logic $\mathcal{L}$, and let $Q$ be a sentence of the form $Q_1 \vee \ldots \vee Q_n$. Assume that $\beta$ and $Q_1, \ldots, Q_n$ are existential sentences of the form*

$$(\exists\, \bar{Y})\, p_1(\bar{Y}_1) \wedge \ldots \wedge p_m(\bar{Y}_m)$$

*where $p_1, \ldots, p_m$ are either roles or concepts appearing in $\mathcal{T}$, $\bar{Y}, \bar{Y}_1, \ldots, \bar{Y}_m$ are tuples of variables and constants, and $\bar{Y} \subseteq \bar{Y}_1 \cup \ldots \cup \bar{Y}_m$. The variables that do not appear existentially quantified in $Q$ or $\beta$ are considered universally quantified. Any variable that appears in $Q$ must also appear in $\beta$.*

*The existential entailment problem is to decide whether*

$$\beta \cup \mathcal{T} \models Q_1 \vee \ldots \vee Q_n. \; \Box$$

The existential entailment problem is important in our context for the following reasons:

- In the case in which $\beta$ is a conjunction of ground atomic facts, existential entailment enables us to deduce whether the disjunction of the antecedents of a set Horn rules is entailed from the ground facts.[3]
- When $n = 1$, existential entailment amounts to subsumption of conjunctive queries over the description logic. In contrast to subsumption of concepts, this problem has not been considered in previous work.

It is important to emphasize that the existential entailment problem *cannot* be reduced to a satisfiability problem of an $\mathcal{ALCNR}$ knowledge base, because the negation of the sentence $Q$ cannot be expressed in an $\mathcal{ALCNR}$ knowledge base. Therefore, the algorithm in [11] does not suffice.

We describe an existential entailment algorithm for the language $\mathcal{ALCNR}$. Our algorithm is based on the technique of *constraint systems,* also used in [11] for a satisfiability checking of $\mathcal{ALCNR}$ knowledge bases and previously in [33, 15]. Informally, in our setting, a constraint system represents a set of models of $\beta \cup \mathcal{T}$. The algorithm begins with an initial constraint system, $S_\beta$, constructed from $\beta \cup \mathcal{T}$. The initial constraint system represents the set of *all* models of $\beta \cup \mathcal{T}$. The algorithm then applies a set of *propagation rules* that generate a set of *completions.* Each completion is a refinement of the initial constraint system, in which some implicit constraints have been made explicit, and some non-deterministic choices have been made. Some completions contain explicit *clashes*, (e.g., they state that an individual belongs both to a class and to its complement), and are then clearly unsatisfiable. Each clash-free completion represents a subset of the models of $\beta \cup \mathcal{T}$, and together they

[3] We will see later that entailing antecedents of CARIN rules is not complicated by the fact that they contain ordinary predicates in addition to concepts and roles.

provide a finite representation of all the models of $\beta \cup \mathcal{T}$. The important property of our completions is that checking whether the formula $Q$ is entailed from a clash-free completion can be done by checking whether the formula is satisfied in one *canonical model* of the completion. Therefore, to check whether $Q$ is entailed by $\beta \cup \mathcal{T}$, it suffices to check the canonical models of each of the clash-free completions. To summarize, our algorithm has four steps:

1. build an initial constraint system $S_\beta$ from $\beta \cup \mathcal{T}$,
2. apply the propagation rules to $S_\beta$ to obtain a set of completions, $\mathcal{S}$,
3. for every clash-free completion $S \in \mathcal{S}$, build a canonical interpretation $I_S$, and
4. check whether $Q$ is satisfied in *all* the canonical interpretations that have been constructed.

The key difficulty in designing the algorithm (and other algorithms based on constraint systems) is to define the termination condition for the second step, in such a way that the resulting completions have the desired properties.

### Constraint Systems

We begin by introducing the elements of constraint systems. Formally, a constraint system is a non-empty set of constraints of the form $s : C$, $sPt$, $\forall x.x : C$, and $s \neq t$, where $C$ is a concept description and $P$ is a primitive role name.

We denote the set of variables and constants that appear in $Q$ or in $\beta$ by $\mathcal{V}$. From this point on, we refer to elements of $\mathcal{V}$ as *individuals*. In describing constraint systems, we introduce a new alphabet of variable symbols $\mathcal{W}$, with a well-founded total ordering $\prec_{\mathcal{W}}$. The alphabet $\mathcal{W}$ is disjoint from $\mathcal{V}$. We denote elements of $\mathcal{W}$ by the letters $u, v, w, x, y, z$. The term *object* refers to elements of $\mathcal{V} \cup \mathcal{W}$ (i.e., either variables or individuals). Objects are denoted by the letters $s, t$. Elements in $\mathcal{V}$ are denoted by the letters $a, b$.

Suppose $S$ is a constraint system and $R$ is a role defined by the description $R = P_1 \sqcap \ldots \sqcap P_k$ ($k \geq 1$). We say that an object $t$ is an $R$-successor of an object $s$ in $S$, if $sP_1t, \ldots, sP_kt \in S$. We say that $t$ is a *direct successor* of $s$ in $S$, if it is the $R$-successor for some role $R$ in $S$. The *successor* relationship denotes the transitive closure of the direct-successor relation. The *direct-predecessor* and the *predecessor* relations are the inverses of direct-successor and successor, respectively. We say that $s$ and $t$ are *separated* in $S$ if $s \neq t \in S$. Finally, we denote by $S[x/s]$ the constraint system obtained from $S$ by replacing each occurrence of the variable $x$ by the object $s$.

For a variable $v$ in a constraint system $S$, we define the function $\sigma(S, v) := \{C \mid v : C \in S\}$. Two variables $v, w \in S$ are said to be *concept-equivalent* if $\sigma(S, v) = \sigma(S, w)$. Intuitively, two variables are concept-equivalent in $S$ if, as far as the constraints in $S$ are concerned, they have the same properties, and therefore, unless they are separated in $S$, they *may* denote the same element in the domain. The function $\sigma$ plays an important role in the termination condition of the algorithm. A constraint system contains a *clash*, if it contains

- $\{s : \perp\}$, or
- $\{s : A, s : \neg A\}$, or
- $\{s : (\leq n\, R)\} \cup \{sP_1t_i, \ldots, sP_kt_i \mid i \in 1..n+1\} \cup \{t_i \neq t_j \mid i, j \in 1..n+1, i \neq j\}$ where $R = P_1 \sqcap \ldots \sqcap P_k$.

The semantics of constraint systems is given by interpretations, mapping each element $s$ of $\mathcal{V} \cup \mathcal{W}$ to an element $\alpha^I(s)$ in the domain $\mathcal{O}^I$, and specifying extensions for the concepts and roles. An interpretation satisfies the constraint $s : C$ if $\alpha^{\mathcal{I}}(s) \in C^I$, the constraint $sRt$ if $(\alpha^I(s), \alpha^I(t)) \in R^I$, the constraint $s \neq t$ if $\alpha^I(s) \neq \alpha^I(t)$, and the constraint $\forall x.x : C$ if $C^I = \mathcal{O}^I$. An interpretation $\mathcal{I}$ is a model of a constraint system $S$ if it satisfies every constraint in $S$. Note that a constraint system with a clash is unsatisfiable.

## 3.1 The Algorithm

We first describe the algorithm for the case in which $\beta$ contains no variables, and therefore, because of the unique names assumption, all individuals in $\beta$ are necessarily mapped to distinct objects in the domain. A minor modification to the algorithm to accommodate variables is described in the end of this section. Without loss of generality, we can suppose that $\mathcal{T}$ contains only inclusion statements and role definitions. We describe the different steps of the algorithm.

### 3.1.1 Creating the Initial Constraint System

We first construct the initial constraint system $S_\beta$ from $\beta \cup \mathcal{T}$. Assume $\beta$ is of the form $p_1(\bar{a}_1) \wedge \ldots \wedge p_n(\bar{a}_n)$. We construct $S_\beta$ as follows.

A1. For every $i \in [1..n]$, if $p_i(\bar{a}_i)$ is of the form $C(a)$, we put $a : C$ in $S_\beta$.

A2. For every $i \in [1..n]$, if $p_i(\bar{a}_i)$ is of the form $R(a, b)$ we put $aP_1b, \ldots, aP_mb$ in $S_\beta$, if $R = P_1 \sqcap \ldots \sqcap P_m$ (or simply $aRb$ if $R$ is a primitive role).

A3. For every inclusion statement $C \sqsubseteq D$ in $\mathcal{T}$, we add $\forall x.x : \neg C \sqcup D$ to $S_\beta$.

A4. For every pair of individuals $a$ and $b$ in $\beta$, we add $a \neq b$ to $S_\beta$.

A5. For every concept $C$ that appears in $Q$, we add the constraint $\forall x.x : C \sqcup \neg C$ to $S_\beta$.

It should be noted that the last set of constraints (which is not added in [11]) is needed because our algorithm is meant to test entailment and not satisfiability. These constraints have the effect that in every completion $S$ that our algorithm generates, $S \models C(s)$ or $S \models \neg C(s)$ for every object $s$, and concept $C$ that appears in $Q$.[4] We assume that all the concepts in a constraint system are *simple,* i.e., the only complements they contain are of

[4] Adding this set of constraints has the same effect of adding the *choose-rule* used in [20, 4], which forces every object in the constraint system to belong either to a concept or to its negation.

the form $\neg A$ where $A$ is the name of a primitive concept. As shown in [15], every $\mathcal{ALCNR}$ concept can be transformed into an equivalent simple concept in linear time.

The following observation follows immediately from the definitions. Intuitively, it shows that $S_\beta$ is an accurate representation of $\beta \cup \mathcal{T}$ in terms of constraint systems. Note that interpretations of $\beta \cup \mathcal{T}$ and of $S_\beta$ are comparable because they have the same object constants and relations.

**Observation 3.1:** An interpretation $\mathcal{I}$ is a model of $\beta \cup \mathcal{T}$ if and only if it is a model of $S_\beta$. $\Box$

### 3.1.2 Applying the Propagation Rules

The next step of the algorithm is to apply the set of propagation rules shown in Figure 1 to the constraint system $S_\beta$ (these are the same propagation rules as in [11]). Informally, each propagation rule corresponds to one of the constructors in $\mathcal{ALCNR}$. For example, rule 3 propagates the constraints implied by the $\forall R.C$ constructor onto a filler, while rule 5 creates additional variables in a constraint system to ensure that the $(\geq\ n\, R)$ constraint is satisfied. Rules 2 and 6 are said to be non-deterministic rules, and all the others are said to be deterministic rules. Two of the propagation rules are *generating* rules because they add new variables to the constraint system.

#### The Termination Condition

A naive application of the propagation rules may not terminate. Therefore, rules 4 and 5 are applied only on variables that are not *blocked*. A key point in designing an algorithm based on constraint systems, and where our use of constraint systems differs from [11], is the definition of blocked variables. To illustrate the subtlety involved, we first illustrate the standard use (e.g, as used in [11]) of constraint systems with an example.

**Example 3.1:** Consider the terminology $\mathcal{T}_2$ consisting of the single inclusion $C \sqsubseteq \exists R.C$, and $\beta_2 = C(a)$. That is, the instances of $C$ must have at least one filler of $R$ that is also an instance of $C$.

Figure 2 shows the application of the propagation rules to this example. The initial constraint system is $S_{\beta_2} = \{a : C, \forall x.x : \neg C \sqcup \exists R.C\}$, whose constraints correspond to $\beta_2$ and the inclusion statement. The first constraint states that $a$ is an instance of $C$, and the second constraint states that every individual is an instance of $\neg C \sqcup \exists R.C$.

The first propagation rule that is applied (rule 7) instantiates the second constraint with the individual $a$. It creates the constraint system $S_1 = S_{\beta_2} \cup \{a : \neg C \sqcup \exists R.C\}$. Because of the disjunction, two successor constraint systems to $S_1$ can be created (using rule 2), one in which $a : \neg C$ is added to $S_1$, and the other in which $a : \exists R.C$ is added. Since the first one is unsatisfiable, we only consider the second one further: $S_2 = S_1 \cup \{a : \exists R.C\}$. The system $S_2$ implies that $a$ has a filler on the role $R$, therefore, a *generating* rule (rule 4) is applied, which adds a new variable $v_1$ to $S_2$ with the appropriate constraints, to obtain

1. $S \rightarrow_{\sqcap} \{s : C_1, s : C_2\} \cup S$
   if 1. $s : C_1 \sqcap C_2$ is in $S$,
   2. $s : C_1$ and $s : C_2$ are not both in $S$.
2. $S \rightarrow_{\sqcup} \{s : D\} \cup S$
   if 1. $s : C_1 \sqcup C_2$ is in $S$,
   2. neither $s : C_1$ nor $s : C_2$ are in $S$,
   3. $D = C_1$ or $D = C_2$.
3. $S \rightarrow_{\forall} \{t : C\} \cup S$
   if 1. $s : \forall R.C$ is in $S$,
   2. $t$ is an $R$-successor of $s$,
   3. $t : C$ is not in $S$.
4. $S \rightarrow_{\exists} \{sP_1y, \ldots, sP_ky, y : C\} \cup S$
   1. $s : \exists R.C$ is in $S$,
   2. $R = P_1 \sqcap \ldots \sqcap P_k$,
   3. $y$ is a new variable,
   4. there is no $t$ such that $t$ is an $R$-successor of $s$ in $S$ and $t : C$ is in $S$,
   5. if $s$ is not blocked.
5. $S \rightarrow_{\geq} \{sP_1y_i, \ldots, sP_ky_i \mid i \in 1..n\} \cup \{y_i \neq y_j \mid i, j \in 1..n, i \neq j\} \cup S$
   if 1. $s : (\geq n\, R)$ is in $S$,
   2. $R = P_1 \sqcap \ldots \sqcap P_k$,
   3. $y_1, \ldots, y_n$ are new variables,
   4. there do not exist $n$ pairwise separated $R$-successors of $s$ in $S$,
   5. if $s$ is not blocked.
6. $S \rightarrow_{\leq} S[y/t]$
   if 1. $s : (\leq n\, R)$ is in $S$,
   2. $s$ has more than $n$ $R$-successors in $S$,
   3. $y, t$ are two $R$-successors of $s$ which are not separated
7. $S \rightarrow_{\forall x} \{s : C\} \cup S$
   if 1. $\forall x.x : C$ is in $S$,
   2. $s$ appears in $S$,
   3. $s : C$ is not in $S$.

Figure 1: Propagation rules.

$S_3 = S_2 \cup \{aRv_1, v_1 : C\}$. Then, the same sequence of propagation rules that were applied to $a$ are also applied to $v_1$, to obtain the following constraint system: $S_4 = S_3 \cup \{v_1 : \neg C \sqcup \exists R.C, v_1 : \exists R.C, v_1Rv_2, v_2 : C\}$. At this point, the same propagation rules *could* be applied to $v_2$. However, $v_1$ and $v_2$ satisfy the same constraints in $S_4$. Hence, $v_2$ is said to be *blocked* by $v_1$, and the propagation terminates.

The canonical model $I$ of $S_4$ which is considered in [11] is built as follows. The domain of $I$ is $\{a, v_1, v_2\}$, and the extensions of the relations are: $C^I = \{a, v_1, v_2\}$, and $R^I = \{(a, v_1), (v_1, v_2), (v_2, v_2)\}$.

The above illustrated the termination condition used in [11], which was designed especially to check satisfiability of an $\mathcal{ALCNR}$ knowledge base. The subtle point in the algorithm is that when building the canonical model, fillers to the blocked variables *have* to be assigned. Above, a filler for role $R$ had to be assigned to $v_2$. In doing so, cycles were introduced in the canonical model that do not exist in *every* model of $S_4$. For example, if the query $Q$ were $\exists x_1, x_2, x_3 R(x_2, x_1) \wedge R(x_3, x_1) \wedge R(x_2, x_3)$, it would be satisfied in the canonical model, even though it is not entailed by $S_4$. However, we want to use the canonical interpretation to check which formulas are satisfied in *all* models of the completion. To avoid this problem, we develop a termination condition that depends on the query, and that guarantees that the resulting canonical model is sufficient for checking the entailment of the query. □

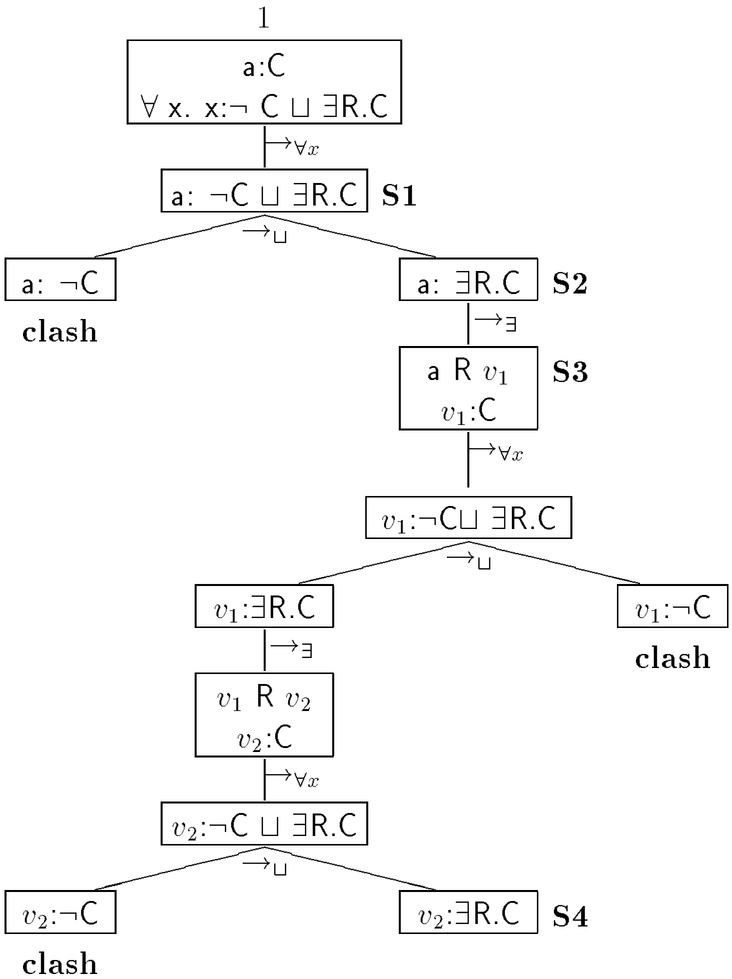


Figure 2: The trace of the application of propagation rules to Example 3.1. In the figure, we only show the constraints that are *added* to the constraint system at every step. Nodes marked with **clash** represent unsatisfiable constraint systems.

Examining the propagation rules reveals that if we only consider the variables in a constraint system, it forms a forest of trees. Specifically, if we consider a graph whose nodes are the variables and there is an arc from a node $x$ to $y$ if $y$ is a direct successor of $x$, then the graph is a forest of trees. This follows from the fact that rules 4 and 5 generate new nodes in the forest, and rule 6 only unifies two successors of the *same* node. The other rules do not add nodes or edges to the graph. We can define the *depth* of a variable in a constraint system to be its depth in the tree to which it belongs. Given this structure, we can define the notion of *n-tree equivalence* among variables in a constraint system.

**Definition 3.2:** *The* n-tree *of a variable $v$ is the tree that includes the variable $v$ and its successors, whose distance from $v$ is at most $n$ direct-successor arcs. We denote the set of variables in the $n$-tree of $v$ by $V_n(v)$.*

*Two variables $v, w \in S$ are said to be $n$-tree equivalent in $S$ if there is an isomorphism $\psi : V_n(v) \to V_n(w)$, such that*

- $\psi(v) = w$,
- *for every $s, t \in V_n(v)$, $sPt \in S$ if and only if $\psi(s)P\psi(t) \in S$, and*
- *for every $s \in V_n(v), \sigma(S, \psi(s)) = \sigma(S, s)$.*

□

Intuitively, two variables are $n$-tree equivalent if the trees of depth $n$ of which they are roots are isomorphic. We denote by $D_Q$ the maximum number of literals in any of the $Q_i$'s.

If there exist two n-tree equivalent variables, $v$ and $w$, such that $w \leq_{\mathcal{W}} v$, then we say that $w$ is a *witness* of $v$. The leaves of the $n$-tree of $v$ will be deemed *blocked*. Figure 3 illustrates the relationship between the witness and the blocked variables.

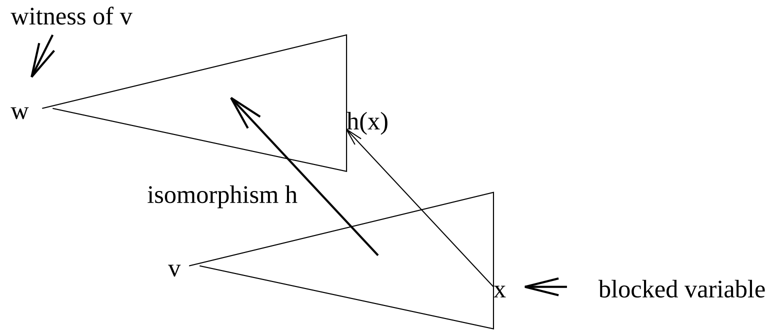


Figure 3: A variable $x$ is blocked if it is the leaf of an $n$-tree rooted at $v$, and $v$ has a witness $w$. $h$ is an isomorphism from the $n$-tree of $v$ to the $n$-tree of $w$. In a canonical interpretation, we will have an *implicit* tuple $(x, t)$, where $t$ is a successor of $h(x)$.

**Definition 3.3:** *(witness) A variable $w$ is a* witness *of a variable $v$ if*

- *$w$ is $D_Q$-tree equivalent to $v$,*
- *$v$ is not in the $D_Q$-tree of $w$, and*
- *there is no other variable $z$, such that $z \prec_{\mathcal{W}} w$, and $z$ satisfies the first two conditions.*

□

**Definition 3.4:** *A variable $x$ is said to be* blocked *if it is a leaf of a $D_Q$-tree whose root is $v$, and $v$ has a witness.* □

In order for the algorithm to detect correctly the blocked variables, the propagation rules are applied according to the following strategy:

- Apply a rule to a variable only if no rule is applicable to an individual.
- Apply a rule to a variable $x$ only if there is no rule applicable to a variable $y$, such that $y \prec_{\mathcal{W}} x$.
- Apply a generating rule only if a non-generating rule cannot be applied.

As in [11], a variable can be deemed blocked *only* after the strategy above permits to apply to it a generating rule, and every variable is blocked by a single witness. It should be noted that once a generating rule has been applied to a variable $v$ in a constraint system $S$, the value of $\sigma(S', v)$ will be the same as $\sigma(S, v)$ in any constraint system resulting from applying propagation rules to $S$ (Lemma 3.2 in [11]). A constraint system is said to be a *completion* when no propagation rule applies to it.

### 3.1.3 Building the Canonical Interpretation

Given a completion $S$, we define its canonical interpretation $I_S$ as follows:

1. $\mathcal{O}^{I_S} := \{s \mid s \text{ is an object in } S\}$.
2. $\alpha^{I_S}(s) := s$
3. For a primitive concept $A$, $s \in A^{I_S}$ if and only if $s : A \in S$.
4. $(s, t) \in R^{I_S}$ if and only if
   (a) $sRt \in S$, or
   (b) $s$ is blocked, $v$ is the root of the $D_Q$-tree of which $s$ is a leaf, $w$ is the witness of $v$, $\psi$ is an isomorphism between the $D_Q$-trees rooted with $v$ and $w$, and $\psi(s)Rt \in S$ (in this case, $s$ would correspond to $x$ in Figure 3).

The extensions of the complex concepts are computed using the equations in Section 2.1. The following lemma is proved in exactly the same way as in [11] (Theorem 3.6), by noting that if $s$ is a blocked variable in a constraint system $S$, then $\sigma(S,s) = \sigma(S,\psi(s))$.

**Lemma 3.2:** *Let $S$ be a clash-free completion of $S_\beta$, and let $I_S$ be its canonical interpretation. Then, $I_S$ is a model of $S$.* $\Box$

We can therefore refer to the canonical interpretation of a clash-free completion as its *canonical model*. We distinguish two kinds of binary tuples in the extensions of the relations in the canonical model. The tuples that are added because of the second clause (i.e., because of $s$ being blocked) are called *implicit* tuples, and the others are called *explicit* tuples. Implicit tuples are put only in the canonical model, and would not necessarily appear in every model of $S$.

### 3.1.4 Checking the Canonical Interpretations

The algorithm returns that $\beta \cup \mathcal{T} \models Q$ if and only if $Q$ is satisfied in the canonical interpretations of all the clash-free completions of $S_\beta$.

**Example 3.2:** Continuing our example, with the query $\exists x_1, x_2, x_3 R(x_2, x_1) \wedge R(x_3, x_1) \wedge R(x_2, x_3)$. Using our definition of blocked variables, the propagation rules would also generate the variables $v_3, \ldots, v_8$ using the same sequence of rule applications that applied to $v_1$ and $v_2$ (see Figure 4). The variable $v_1$ is then recognized as the witness of $v_5$. The variable $v_8$ would be deemed blocked, because it is the leaf of a 3-tree rooted in $v_5$, and the 3-trees rooted in $v_1$ and $v_5$ are isomorphic. Therefore, the canonical model of the completion would have the implicit tuple $(v_8, v_5)$ in $R^{I_S}$ (because $\psi(v_8) = v_4$ and $v_4 R v_5 \in S$). The query $Q$ is not satisfied in the canonical interpretation, and therefore is not entailed by $\mathcal{T}_2 \cup \beta_2$. $\Box$

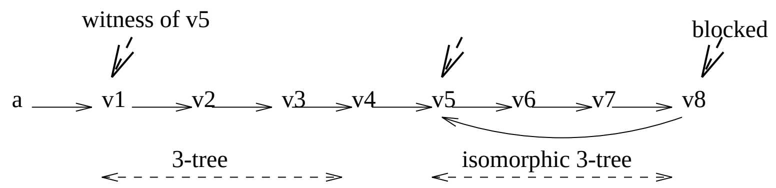


Figure 4: The application of the propagation rules with our termination condition to Example 3.1. In this example, we consider 3-trees. The 3-tree of $v_5$ is isomorphic to the 3-tree of $v_1$. Therefore, the variable $v_8$ (which is a leaf of the 3-tree of $v_5$) is blocked. In the canonical interpretation we have the implicit tuple $(v_8, v_5)$.

## Allowing Variables in $\beta$

Recall that so far we have assumed that $\beta$ does not contain any variables, and hence, when we create $S_\beta$ from $\beta$ we make use of the unique names assumption. Therefore $S_\beta$ only represents those models of $\beta$ in which each constant is mapped to a distinct object in the interpretation. When $\beta$ contains variables, $\beta$ may have models in which two variables are mapped to the same object in the domain. To check entailment in this case we need to apply the algorithm to any homomorphism $h$ on $\beta$. A homomorphism $h$ maps the variables of $\beta$ to variables or constants appearing in $\beta$. If $Q$ is entailed from $S_{h(\beta)} \cup \mathcal{T}$ for any homomorphism $h$, then $Q$ is entailed from $S_\beta \cup \mathcal{T}$.

## 3.2 Proof of Correctness and Complexity

In this section we prove the correctness of the existential entailment algorithm and discuss its complexity. The structure of the proof is as follows. Theorem 3.3 establishes the basic property of our termination condition and of the resulting completions. It shows that to check that a formula is entailed from a completion, it suffices to check that the formula is satisfied in the canonical interpretation of the completion. Next, we show that a formula is entailed from $\beta \cup \mathcal{T}$ if and only if it is entailed from all the clash-free completions of $S_\beta$. Finally, we show that the application of the propagation rules terminates. For clarity, the more tedious proofs are postponed to the appendix.

**Theorem 3.3:** *Let $S$ be a clash-free completion of $S_\beta$, and let $I_S$ be its canonical model. If $I_S \models Q$, then $S \models Q$.* $\Box$

**Proof:** Since $I_S \models Q$, and $Q = Q_1 \vee \ldots \vee Q_n$, one of the $Q_i$'s must be satisfied in $I_S$. We can assume without loss of generality that it is $Q_1$. Suppose $Q_1$ is of the form

$$(\exists\, \bar{X})\, l_1(\bar{X}_1) \wedge \ldots \wedge l_m(\bar{X}_m).$$

Given a variable $x$ in $Q_1$, we define its *connected component* $V_x$ as the minimal set of variables that satisfy the following conditions:

- $x \in V_x$,
- If $y$ and $z$ appear in the same role atom in $Q_1$, and $z \in V_x$, then $y \in V_x$.

We denote by $L_x$ the set of literals in $Q_1$ that include variables in $V_x$. Since $Q_1$ is satisfied in $I_S$, there must be a mapping $\sigma_0$ from the variables of $Q_1$ to $\mathcal{O}^{I_S}$, such that for every $i$, $1 \le i \le m$, $\sigma_0(\bar{X}_i) \in l_i^{I_S}$.

Our proof proceeds in two steps:

1. First we show that we can modify $\sigma_0$ to a mapping $\sigma$, such that $\sigma(\bar{X}_i) \in l_i^{I_S}$, and $\sigma(\bar{X}_i)$ is an *explicit* tuple in $l_i^{I_S}$, for every $i$, $1 \le i \le m$. Recall that the explicit tuples are the ones that *must* be in the canonical interpretation (given the mapping of the objects, $\alpha^{I_S}$) in order for the canonical interpretation to be a model, while the choice of the implicit tuples is, to some extent, arbitrary.
2. In the second step, we use the mapping $\sigma$ to show that $Q_1$ will be satisfied in *every* model of $S$.

**Step 1:** If $\sigma_0$ makes use of implicit tuples in $I_S$, then there must be some variables $x$ in $Q_1$ such that $\sigma_0(x)$ is a blocked variable in $S$. For each such variable $x$, let us consider each connected component $V_x$ of $x$ in $Q_1$. We first note the following properties.

1. *There is no variable $z \in V_x$, such that $\sigma_0(z)$ is an individual in $S$.*

   If this were not true, it would entail that there is a chain of direct-successor arcs of length at most $D_Q - 1$ from an individual in $S$ to a blocked variable in $S$. Such a chain cannot exist because every blocked variable is the leaf of a $D_Q$-tree whose root is a variable in $S$.

2. Let $G_x$ be the graph whose nodes are variables of $V_x$, and which contains an arc from $y$ to $z$ if there is a atom $R(y, z)$ in $L_x$. Let $\sigma_0(G_x)$ denote the image of $G_x$ under the mapping $\sigma_0$:

   *There is no cycle in $\sigma_0(G_x)$ and therefore there is no cycle in $G_x$.*

   Since all variables in $V_x$ are mapped by $\sigma_0$ to variables in $S$, the only way that there can be a cycle in $\sigma_0(G_x)$ is if it were caused by an implicit tuple in $I_S$. By definition, an implicit link goes from a blocked variable $s$ to a variable $t$ such that

   - $s$ is a leaf of a $D_Q$ whose root is $v$,
   - $v$ has a witness $w$, and
   - the predecessor of $t$ is a leaf in the $D_Q$-tree rooted in $w$.

   Since Definition 3.4 requires that a variable $v$ *not* be in the $D_Q$ tree of its witness, it follows that the distance between $t$ and $s$ is at least the distance between $s$ and $v$. Hence, the distance between $s$ and $t$ is at least $D_Q$, and therefore, since $\sigma_0(G_x)$ and $G_x$ have at most $D_Q$ edges, they cannot contain a cycle (see Figure 3).

Let $x_1, \ldots, x_k$ be a topological ordering on the variables of $G_x$, (which is well defined because $G_x$ is acyclic). We consider a sequence of variable mappings from $G_x$ to $\mathcal{O}^{I_S}$, $\sigma_0, \sigma_1, \ldots, \sigma_k$. For every mapping we show that:

- $\sigma_i$ is a satisfaction mapping for $Q_1$, i.e., for every $j$, $1 \leq j \leq m$, $\sigma_i(\bar{Y}_j) \in l_j^{I_S}$, and
- the only possible implicit tuples in $\sigma_i(G_x)$ emanate from the $\sigma_i(x_i), \ldots, \sigma_i(x_k)$.

The desired mapping $\sigma$ is simply defined to be $\sigma_k$. Note that for $\sigma_0$, the properties hold trivially. We show how to construct $\sigma_{i+1}$ from $\sigma_i$.

Let $y$ be lowest variable in the topological ordering of $G_x$, such that $\sigma_i(y)$ is blocked in $S$. We denote by $\mathcal{Y}$ the set of variables that are the leaves of the same $D_Q$-tree of which $y$ is a leaf. We can distinguish three sets of links in $\sigma_i(G_x)$:

A1. links emanating from ancestors of variables in $\mathcal{Y}$,

A2. links emanating from variables in $\mathcal{Y}$, and

A3. links between descendents of variables in $\mathcal{Y}$.

Let $v$ be the root of the $D_Q$-tree whose leaf is $y$, and let $\psi$ be the isomorphism between $v$ and its witness. We define $\sigma_{i+1}$ as follows:

B1. for a variable $z$ that is either in $\mathcal{Y}$ or is an ancestor of a variable in $\mathcal{Y}$, $\sigma_{i+1}(z) = \psi(\sigma_i(z))$

B2. for the other variables $\sigma_{i+1}$ is identical to $\sigma_i$.

First, we show that $\sigma_{i+1}$ is a satisfaction mapping for $Q_1$. Consider two cases for a conjunct $l_j(\bar{Y}_j)$ of $Q_1$:

- if $l_j$ is a concept, then in case B1, since $\psi$ conserves concept equivalence, it guarantees that $\sigma_i(\bar{Y}_j) \in l_j^{I_S}$ if and only if $\sigma_{i+1}(\bar{Y}_j) \in l_j^{I_S}$. In B2 the claim holds trivially.
- if $l_j$ is a role atom, then case A3 is trivial since $\sigma_{i+1}$ does not affect the descendents of variables in $\mathcal{Y}$. Case A1 follows from the definition of $n$-tree equivalence. Case A2 is the one in which $\sigma_i(\bar{Y}_j)$ is an implicit tuple in $I_S$ of the form $(s,t)$, where $(\psi(s),t)$ is an explicit tuple in $I_S$. By the construction of $\sigma_{i+1}$ it follows that $\sigma_{i+1}(\bar{Y}_j)$ is $(\psi(s),t)$, and therefore the claim follows.

Finally, note that for any variable $z$ in $\mathcal{Y}$ or that is an ancestor of a variable in $\mathcal{Y}$, the tuples of the form $(\sigma_{i+1}(z),t)$ are *explicit* tuples in $I_S$. Hence, by a simple induction on $i$ it can be shown that the only possible implicit tuples in $\sigma_{i+1}(G_x)$ emanate from the $\sigma_{i+1}(x_{i+1}),\ldots,\sigma_{i+1}(x_k)$. $\square$

**Step 2:** Let $I$ be a model of $S$. To complete the proof we need to show that $I \models Q_1$. To do that, we need to find a mapping $\phi$ from the variables of $Q_1$ to the domain of $I$, such that $\phi(\bar{X}_i) \in l_i^I$ for every $i$, $1 \le i \le m$.

Since $I$ is a model of $S$, it means that there exists a mapping $\alpha$ from the objects of $S$ to the domain of $I$ such that if $v : C \in S$, then $\alpha^I(v) \in C^I$, and if $sRt \in S$, then $(\alpha^I(s), \alpha^I(t)) \in R^I$. Let the mapping $\phi$ be defined by $\phi(x) = \alpha^I(\sigma(x))$. We show that for every $i$, $1 \le i \le m$, $\phi(\bar{X}_i) \in l_i^I$. There are two cases:

- The literal $l_i(\bar{X}_i)$ is unary, i.e., of the form $C(x)$. In this case, $\sigma(x) \in C^{I_S}$. Recall that $s : C \in S$ or $s : \neg C \in S$ for any concept $C$ appearing in $Q$, and therefore, it must be the case that $\sigma(x) : C \in S$. Therefore, since $I$ is a model of $S$, it follows that $\alpha^I(\sigma(x)) \in l_i^I$.
- The second case is when the literal $l_i(\bar{X}_i)$ is of the form $R(x,y)$. In this case, $(\sigma(x),\sigma(y))$ is an *explicit* tuple of the canonical model $I_S$, which means that $\sigma(x) R \sigma(y) \in S$, and therefore, $(\alpha^I(\sigma(x)), \alpha^I(\sigma(y))) \in R^I$.

$\square$

The next step in proving the correctness of our algorithm is to show that the union of the clash-free completions of $S_\beta$ is equivalent to $S_\beta$. This is formalized by the following lemma.

**Lemma 3.4:** *Let $S$ be a constraint system, and let $\mathcal{S}$ be the set of all possible constraint systems obtainable from $S$ using a propagation rule $r$ (if $r$ is a deterministic propagation rule, then $\mathcal{S}$ is a singleton set). Then, $S \models Q$ if and only if, for all $S' \in \mathcal{S}$, $S' \models Q$* $\Box$

The proof of the lemma appears in the appendix. The following corollary shows that entailment of $Q$ can be checked by verifying entailment in each of the clash-free completions of $S_\beta$. The corollary follows by iteratively applying Lemma 3.4.

**Corollary 3.5:** *Let $S$ be a constraint system, and let $\mathcal{S}$ be the set of all its clash-free completions. Then, $S \models Q$ if and only if for every $S' \in \mathcal{S}$, $S' \models Q$.* $\Box$

The following lemma shows that our algorithm always terminates.

**Lemma 3.6:** *Given the constraint system $S_\beta$, the application of the propagation rules will terminate.* $\Box$

**Proof:** The number of times that a propagation rule can be applied to an object in a constraint system is bounded by the size of the terminology. Each application of a propagation rule adds a number of successor variables which is also bounded by the size of the terminology (specifically, the largest number appearing in the number restrictions). Therefore, in order to show that the algorithm terminates, it suffices to show that there is a bound on the *depth* of the variables in a constraint system.

To bound the depth, we show that there is a finite number $N$ of non isomorphic $D_Q$-trees. Hence, the depth of a variable in a constraint system is bounded by $(N+1)D_Q$. To see why, suppose there would be a variable $v$ of depth $(N+1)D_Q$ with a successor $w$. The variable $v$ would be a leaf of a sequence of $N+1$ non-overlapping $D_Q$-trees. Therefore, there would definitely be two of them that are $D_Q$-tree equivalent, and therefore, either $v$ or one of its ancestors would be blocked, and $w$ would not be generated. We now derive a bound on the value of $N$. We use the following notation:

- $B$ denotes the size of $S_\beta$, i.e., the sum of the sizes of the constraints in $S_\beta$.
- $C_\beta$ denotes the set concept descriptions that appear in either in $S_\beta$ or as subdescriptions of concept descriptions in $S_\beta$, and $C$ denotes the cardinality of the set $C_\beta$. Note that $C$ is linear in the size of $S_\beta$.
- $r$ denotes the number of different role names in $C_\beta$, and $r_{max}$ denotes the maximal number appearing in the number restrictions in $S_\beta$ (and is 1 if there are no number restrictions or if only 0 appears in $S_\beta$).

We derive a (generous) upper bound on the maximal number of non-isomorphic $n$-trees, denoted by $q_n$. The value of $N$ is $q_{D_Q}$.

For $n = 0$, the number of different $n$-trees is at most $2^C$, corresponding to the different possible values for the function $\sigma(S, v)$.

Consider the number of possible $n$-trees for $n > 0$. The root $v$ of such a tree has at most $2^C$ different possible values for the $\sigma(S, v)$. Consider a single role $R$. The root $v$ can have at most $Cr_{max}$ direct successors of $R$. This is because the number of times a generating rule can be applied to $v$ is at most $C$ and each application can add at most $r_{max}$ successors. Each successor can be the root of an $(n-1)$-tree. Hence, there are $O(2^C q_{n-1}^m)$ non isomorphic subtrees in the case when $v$ has exactly $m$ direct successors of $R$ and no other direct successors.

Since there are $(Cr_{max}) + 1$ possible choices for the number of direct successors of $R$ for $v$, and each choice is bounded by $Cr_{max}$, then, if we consider a single role, there are $O(2^C Cr_{max} q_{n-1}^{Cr_{max}})$ non isomorphic $n$-trees rooted at $v$.

Finally, since we can repeat the above process for every role $R$ in $S_\beta$, and there are $r$ roles, we obtain

$$q_n = O(2^C (Cr_{max} q_{n-1}^{Cr_{max}})^r).$$

To simplify notation, let $x = 2^C (Cr_{max})^r$ and $a = rCr_{max}$. The equation can be rewritten as:

$$q_n = O(x(q_{n-1})^a)$$

Unfolding the equation yields:

$$q_n = O((x^{1+a \ldots + a^{n-1}})(q_0)^{a^n}) = O((x2^C)^{a^n}).$$

Returning to our original notation, we obtain

$$q_n = O((2^{2C}(Cr_{max})^r)^{(rCr_{max})^n}$$

□

The following theorem establishes the correctness of our algorithm.

**Theorem 3.7:** *$\mathcal{T} \cup \beta \models Q$ holds if and only if $Q$ is satisfied in the canonical interpretation of every clash-free completion of $S_\beta$.* □

**Proof:** *only if:* Suppose $S$ is a clash-free completion of $S_\beta$ and in the canonical interpretation $I_S$ of $S$, $I_S \not\models Q$. Since, by Lemma 3.2, $I_S$ is a model of $S$, it follows that $S \not\models Q$. Therefore, by Corollary 3.5 and Observation 3.1, $\beta \cup \mathcal{T} \not\models Q$.

*if:* Let $\mathcal{S}$ be the set of clash-free completions of $S_\beta$, and suppose that for every $S \in \mathcal{S}$, the canonical interpretation $I_S$ of $S$ satisfies $Q$. By Theorem 3.3 it follows that $S \models Q$ for every $S \in \mathcal{S}$. By Corollary 3.5 it follows that $S_\beta \models Q$, and therefore, by Observation 3.1, it follows that $\beta \cup \mathcal{T} \models Q$. □

### 3.2.1 Complexity

The proof of Lemma 3.6 shows that the number of non-isomorphic $D_Q$-trees is at most doubly exponential in the size of the $\beta \cup \mathcal{T}$. The number of variables in a $D_Q$-tree is also at most doubly exponential in the size of $\beta \cup \mathcal{T}$, because the depth of the tree is $D_Q$ and the branching factor of the tree is possibly exponential in the size of $\beta \cup \mathcal{T}$ (note that the branching factor is exponential because we are assuming that the numbers in the number restrictions are encoded in binary form).

In the worst case, a constraint system will contain an $n$-tree for every $n$-tree equivalence class and for each such tree, all the leaves will have $n$-trees as well. Hence, the maximal number of objects in a completion of $S_\beta$ is at most doubly exponential in the size of $\beta \cup \mathcal{T}$.

Checking that the sentence $Q$ is satisfied in a canonical interpretation takes time at most exponential in the size of $Q$, and polynomial in the size of the canonical interpretation (this is the time needed to evaluate a conjunctive query over a database).

In a similar way to what was shown in [11], it can be shown that the time to construct a completion is doubly exponential in the size of $\beta \cup \mathcal{T}$. Consequently, the time complexity of checking that $\beta \cup \mathcal{T} \not\models Q$ is non-deterministic doubly exponential in the size of $\beta \cup \mathcal{T}$, and triply exponential in the size of $\beta \cup \mathcal{T} \cup Q$.

It is important to emphasize that the source of one exponent in the complexity analysis is the fact that the numbers appearing in the number restrictions in $\beta \cup \mathcal{T}$ are encoded in binary form. If we assume a unary encoding of numbers, or if we assume that the magnitude the numbers is bounded by the size of $\beta \cup \mathcal{T}$ (which is very likely if $\beta$ are the ground atomic facts in the knowledge base, as we see in the next section), then the time-complexity of determining $\beta \cup \mathcal{T} \not\models Q$ is non-deterministic exponential time. The time complexity of the existential entailment problem in this case is deterministic doubly exponential time (the entailment problem requires checking all completions, and there are a doubly exponential number of completions). The complement of the existential entailment problem we consider is at least as hard as the KB-satisfiability problem considered in [11]. The algorithm given there has a time complexity of non-deterministic exponential time, though they assume that numbers are encoded in binary form.

It should be noted that it is not always necessary to apply the existential entailment algorithm based on $D_Q$-tree equivalence of variables. In fact, the only problem that the canonical interpretation $I_S$ can introduce is to contain a cycle that does not exist in every model of $S$. We can view each $Q_i$ as a graph, where the nodes are the variables of $Q_i$ and there is an arc from $x$ to $y$ if there is a literal of the form $R(x, y)$ in $Q_i$. We can consider $N_{Q_i}$ to be the length of the longest path (without node repetition) that exists in $Q_i$. The existential entailment algorithm can be applied using the $N$-tree equivalence relation, where $N$ is the maximal $N_{Q_i}$ for $Q_i \in Q$.[5] Finally, we note that we have provided only a worst-case complexity analysis, and the issue of optimization is beyond the scope of this paper.

# 4 Uses of the Existential Entailment Algorithm

The existential entailment algorithm is a key tool for reasoning in CARIN. In this section we describe two important uses of existential entailment. First, in Section 4.1 we show that the existential entailment algorithm provides the basis for a sound and complete reasoning algorithm for nonrecursive CARIN-$\mathcal{ALCNR}$ knowledge bases. In Section 4.2 we show that

[5] We thank Werner Nutt for pointing out this optimization to us.

the existential entailment algorithm also provides a sound and complete algorithm for query containment over $\mathcal{ALCNR}$.

## 4.1 Sound and Complete Reasoning for Non Recursive CARIN-$\mathcal{ALCNR}$

When Horn rules are not recursive, they are equivalent to a union of conjunctive queries. Given this equivalence, we can develop a sound and complete inference procedure for non recursive CARIN-$\mathcal{ALCNR}$ based on the existential entailment algorithm. In this section we describe our algorithm in detail. It should be noted that the algorithm does *not* actually unfold the rules for performing reasoning.

**Rule Unfolding:** Given a set of Horn rules, we can iteratively *unfold* them. In an unfolding step of a rule $r$ we consider a conjunct in the antecedent of the form $p(\bar{X})$, and a rule $r_1$ whose consequent is of the form $p(\bar{Y})$. We consider the most general unifier $\theta$ of $p(\bar{X})$ and $p(\bar{Y})$, i.e., $\theta(p(\bar{Y})) = \theta(p(\bar{X}))$. The result of the unfolding step is the rule in which the antecedent is $\theta(ant(r_1)) \cup \theta(ant(r)) \setminus \theta(p(\bar{X}))$, where $ant(r)$ denotes the antecedent of a rule $r$. The consequent of the rule is $\theta(h)$, where $h$ is the consequent of $r$. Note that if a variable $v$ appears in the antecedent of $r_1$ and not in its head, then $\theta$ will map it to a fresh variable that appears nowhere else. When the Horn rules are not recursive, the process of unfolding will terminate.

**Example 4.1:** If we have the rules:

$r_1$ : p(X) $\wedge$ p(Y) $\Rightarrow$ r(X,Y)
$r_2$ : e(X,Y) $\wedge$ d(Y) $\Rightarrow$ p(X)

then one step of unfolding $r_1$ can result in the rule

$r'$ : e(X,Z) $\wedge$ d(Z) $\wedge$ p(Y) $\Rightarrow$ r(X,Y). $\Box$

Given the Horn rules in a CARIN-$\mathcal{ALCNR}$ knowledge base $\Delta$, we denote by $U(\Delta)$ the *maximal* size (i.e., number of conjuncts) of a rule that can be obtained by unfolding rules in $\Delta$. Note that $U(\Delta)$ may be exponential in the depth[6] of rules in $\Delta$. Note that the rules do not actually have to be unfolded in order to determine $U(\Delta)$.

Given a knowledge base $\Delta$ and a query $p(\bar{a})$, the algorithm for reasoning in non recursive CARIN-$\mathcal{ALCNR}$ proceeds in two steps:

[6]the depth of a set of rules is the maximal derivation depth of the literals appearing in the rules. Let $q(\bar{X})$ a literal: if $q$ is a base predicate, depth $(q(\bar{X})) = 0$ ; if $q$ is not a base predicate, it appears as a consequent of some rules, let $p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n)$ the literals appearing in the antecedent of those rules: depth$(q(\bar{X})) = 1 + Max\{depth(p_1(\bar{X}_1)), \ldots, depth(p_1(\bar{X}_n))\}$

1. We let $\beta$ be the set of ground facts in $\Delta$ of role and concept predicates, $\beta_r$ be the set of ground facts of ordinary predicates in $\Delta$, and $\Delta_r$ be the set of Horn rules in $\Delta$. We apply the propagation rules to $S_\beta$ with $U(\Delta)$-tree equivalence as the termination condition. Note that $S_\beta$ contains $\forall x.x : C \sqcup \neg C$ for every concept predicate appearing in the rules.

2. In each completion $S$, we compute extensions of the ordinary predicates by evaluating the Horn rules of $\Delta$ using a traditional Horn rule reasoning algorithm. We pay special attention to how we perform lookups. If we are performing a lookup for a fact of the form $C(a)$ where $C$ is a concept description (i.e., trying to determine whether $S \models C(a)$), we check whether $a^{I_S} \in C^{I_S}$. Lookups for role atoms are done similarly. Lookups for atoms of ordinary predicates are done from $\beta_r$. A bottom-up procedure for evaluating the Horn rules is shown in Figure 5.

**procedure horn-evaluate**($\Delta$, $I_S$)
/* $\Delta$ is a CARIN-$\mathcal{ALCNR}$ KB, and $I_S$ is a canonical interpretation of a completion of $S_\beta$. */
/* The procedure computes $e^{I_S}$ for the ordinary predicates $e \in \Delta$. */

Extend the domain $\mathcal{O}^{I_S}$ to include all the constants that appear in $\Delta$.
**for** every ordinary predicate $e \in \Delta$, $e^{I_S} = \{\bar{a} \mid e(\bar{a}) \in \beta_r\}$.

**while** new tuples are being added to the extensions **do:**
  Let $r$ be a Horn rule in $\Delta_r$ of the form $l_1(\bar{X}_1) \wedge \ldots \wedge l_m(\bar{X}_m) \Rightarrow q(\bar{X})$.
    Let $\psi$ be a mapping from the variables of $r$ to $\mathcal{O}^{I_S}$.
    **if** the following holds for $i$, $1 \leq i \leq m$, **then** add $\psi(\bar{X})$ to $q^{I_S}$:
      **if** $l_i$ is an ordinary predicate and $\psi(\bar{X}_i) \in l_i^{I_S}$, or
      **if** $l_i$ is a role predicate $R$, such that $R = P_1 \sqcap \ldots \sqcap P_k$ and $\psi(\bar{X}_i) \in P_j^{I_S}$ for $1 \leq j \leq k$, or
      **if** $l_i$ is a concept predicate and $\psi(X_i) \in l_i^{I_S}$.

**return** the extensions of the ordinary predicates of $\Delta$.
**end.**

Figure 5: An algorithm for bottom-up evaluation of Horn rules in a given completion.

A query of the form $p(\bar{a})$ is entailed by $\Delta$ if and only if $\bar{a}$ is in the extension of $p$ for every clash-free completion $S$. Formally, our algorithm entails the following result:

**Theorem 4.1:** *Let $\Delta$ be a* CARIN-$\mathcal{ALCNR}$ *knowledge base whose Horn rules are not recursive. Let $p(\bar{a})$ be a ground atomic query, where $p$ is either a concept, role or ordinary predicate. The problem of determining whether $\Delta \models p(\bar{a})$ is decidable.* $\Box$

The complexity of the decision procedure is the same as the complexity of the existential entailment algorithm because the evaluation of the Horn rules in each completion takes time that is polynomial in the size of the completion. It is interesting to note that complexity of entailment algorithm is in Co-NP, as a function of the number of ground facts in $\Delta$. This follows because the size of each completion is linear in the number of ground facts in $\Delta$, and checking whether $q(\bar{a})$ is entailed by a canonical model can be done in time polynomial in the size of the completion.

**Proof:** Given a clash-free completion $S$, and its canonical interpretation $I_S$, the process of computing the extension of the ordinary predicates effectively computes an interpretation for $\Delta$. We denote this interpretation by $I_\Delta^S$. It is easy to see that since $I_S$ is a model of $\beta \cup \mathcal{T}$, then $I_\Delta^S$ is a model of all of $\Delta$.

We first prove the completeness of the algorithm. Suppose there exists a clash-free completion $S$ of $S_\beta$ such that in the evaluation of the Horn rules, we do not compute $a$ is not in the extension of $p$ in $I_\Delta^S$. Since $I_\Delta^S$ is a model of $\Delta$, it follows that $\Delta \not\models p(\bar{a})$.

The soundness of our algorithm is a consequence of the following three claims:

1. The existential entailment algorithm described in Section 3 can be extended trivially to the case in which $\beta$ is a set of ground facts, and both $Q$ and $\beta$ may contain ground facts of ordinary predicates (of any arity). Note that such ground facts do not play any role in the propagation phase. Hence, since the termination condition of the propagation phase of our algorithm was based on $U(\Delta)$-tree equivalence, it follows that if $I_S$ is a canonical interpretation, then $I_S \cup \beta_r$ can be used for correctly checking satisfaction of existential sentences with $U(\Delta)$ atoms or less, even if they contain atoms of ordinary predicates.

2. In a similar fashion to the proof of Lemma 3.4 and using Observation 3.1, it can be shown that for a ground fact $p(\bar{a})$,

$$\Delta \models p(\bar{a}) \Leftrightarrow \Delta_r \cup S \models p(\bar{a}) \text{ for every clash-free completion } S \text{ of } S_\beta.$$

3. Finally, we need to show that our procedure for computing the extensions of the ordinary predicates from a canonical interpretation $I_S$ and $\beta_r$ has the property that $\bar{a}$ is in the extension of $p$ if and only if $S \cup \Delta_r \models p(\bar{a})$.

   Suppose that we derived that $\bar{a}$ is in the extension of $p$ from $I_S \cup \beta_r$. In this case, $p(\bar{a})$ has a *derivation tree* $d$. A derivation tree has $p(\bar{a})$ as its root, and its child $r_0$ is the instantiation of a rule in $r \in \Delta_r$ that was used in the final step of deriving $p(\bar{a})$. The children of $r_0$ are the atoms in its antecedent, and their children are the rules used to derive them, etc. The leaves of the tree are either ground atomic facts of ordinary predicates in $\Delta_g$, or concept or role atoms that were looked up in $I_S$. The number of leaves in the tree is at most $U(\Delta)$. Given the tree, it is possible to construct *one* rule $r_1$ with the following properties:

   D1. $r_1$ is the result of unfolding rules in $\Delta_r$, and hence $\Delta_r \models r_1$,

   D2. there is a mapping $\tau$ from the variables of $r_1$ to the constants in $\Delta$, such that $\tau$ maps the consequent of $r_1$ to $p(\bar{a})$, and the atoms in the antecedent of $r_1$ to facts in $I_S \cup \beta_r$,

   D3. the number of atoms in the antecedent of $r_1$ is at most $U(\Delta)$.

Because of D2, the existential entailment algorithm would entail the antecedent of $r_1$ from $I_S \cup \beta_r$. Because of D3, it follows that the antecedent of $r_1$ is entailed by $S \cup \beta_r$. Hence, because of D1, it follows that $\Delta_r \cup S \cup \beta_r \models p(\bar{a})$. □

**Remark 4.2:** An important consequence of Theorem 4.1 is that we obtain the first algorithm for answering arbitrary conjunctive queries (with existentially quantified variables) from an $\mathcal{ALCNR}$ knowledge base. In contrast, previous algorithms only considered answering membership and subsumption queries. We note that conjunctive queries are the basis underlying database query languages such as SQL. □

**Example 4.2:** We illustrate the algorithm on the rules and ground facts given in Section 2. Note that in this case the predicates made-by and monopoly are ordinary predicates.

$r_1$ : made-by(X,Y) $\wedge$ no-fellow-company(Y) $\Rightarrow$ price(X,usa,high)
$r_2$ : made-by(X,Y) $\wedge$ associate(Y,Z) $\wedge$ american(Z) $\wedge$ monopoly(Y,X,usa) $\Rightarrow$ price(X,usa,high).

$\mathcal{A}_2$ : {made-by(a,b),monopoly(b,a,usa), international-company(b)}

Recall that the terminology contains the definition:

no-fellow-company := $\forall$associate.$\neg$ american
international-company := european-associate $\sqcup$ american-associate.

Figure 6 shows the trace of the application of propagation rules to the initial constraint system. The constraints in each node are the ones that are added to the parent constraint system as a result of applying the propagation rule. For clarity, we only show the constraints that are important for our explanation. The first constraint in node 1 comes from the ground fact international-company(b), and the second constraint comes from the instantiation of rule 7 to the concept no-fellow-company $\sqcup$ $\neg$no-fellow-company. In applying the propagation rules, the disjunction rule (rule 2) is first applied to b and the top disjunction in node 1, yielding nodes 2 and 3. In node 2 the disjunction rule is applied again to b the second disjunction in node 1. In node 4 we apply the existential rule to produce v2 and then the universal-quantification rule to assert v2:$\neg$american. In node 5 we apply the existential rule twice, once to produce v3 and once for v4.

Now we compute the extensions of the ordinary predicates in the different completions. In this case $\beta_r$ contains the facts made-by(a,b) and monopoly(b,a,usa). In node 7, no-fellow-company(b) is satisfied and therefore price(a,usa,high) is derivable by rule $r_1$. In the completions of nodes 8 and 6 b has an American associate, and therefore price(a,usa,high) is derivable by rule $r_2$. □

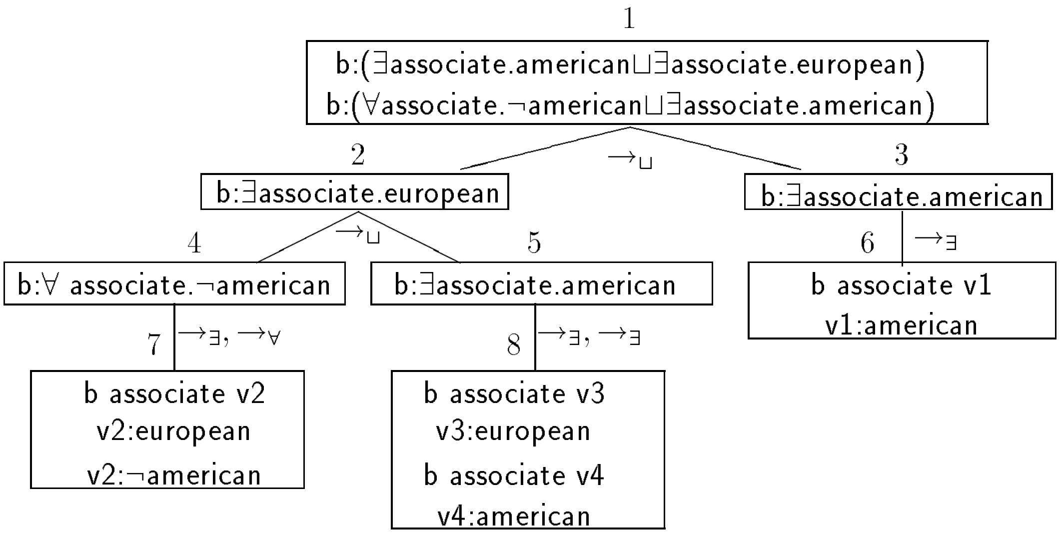


Figure 6: The trace of the application of propagation rules to the initial constraint system.

## 4.2 Query Containment over $\mathcal{ALCNR}$

The second important usage of the existential entailment algorithm is to provide the first sound and complete algorithm for containment of conjunctive queries over $\mathcal{ALCNR}$. In database systems, algorithms for query containment play an important role in several query optimization techniques [32] and related problems (e.g., rewriting queries using views [21], semantic query optimization [10, 27], detecting independence of queries from updates [26]). Therefore, extending these algorithms for conjunctive queries over description logics enables to extend optimization techniques to a setting involving description logics. In particular, [6] uses our query containment algorithm to extend algorithms for rewriting queries using views to views and queries expressed in description logics.

Formally, a conjunctive query over an $\mathcal{ALCNR}$ terminology $\mathcal{T}$ is an expression of the form

$$(\exists\, \bar{Y})\, p_1(\bar{Y}_1) \wedge \ldots \wedge p_m(\bar{Y}_m),$$

where the $p_i$'s are either concepts or role predicates that appear in $\mathcal{T}$. The tuples $\bar{Y}, \bar{Y}_1, \ldots, \bar{Y}_m$ are tuples of variables and constants, and $\bar{Y} \subset \bar{Y}_1 \cup \ldots \cup \bar{Y}_m$. The *distinguished variables* $\bar{X} = X_1, \ldots, X_n$ of a conjunctive query are the variables that do not appear in $\bar{Y}$. Given a set of ground atomic facts, $G$, for concept and role predicates, the answer to the conjunctive query from $G \cup \mathcal{T}$ is any tuple of the form $a_1, \ldots, a_n$, such that

$$G \cup \mathcal{T} \models (\exists\, \bar{Y})\, p_1(\psi(\bar{Y}_1)) \wedge \ldots \wedge p_m(\psi(\bar{Y}_m)),$$

where $\psi$ maps $X_i$ to $a_i$.

**Definition 4.1:** *Let $Q_1$ and $Q_2$ be two conjunctive queries over an $\mathcal{ALCNR}$ terminology $\mathcal{T}$ with the same number of distinguished variables. The query $Q_1$ is contained in $Q_2$ if, for* *any set of ground facts $G$ for the concept and role predicates in $\mathcal{T}$, the set of answers for $Q_1$ from $G \cup \mathcal{T}$ is a subset of the answers for $Q_2$ from $G \cup \mathcal{T}$.* $\Box$

The following theorem follows from the existential entailment algorithm by noting that $\mathcal{T} \cup Q_1 \models Q_2$ if and only if $Q_1$ is contained $Q_2$.

**Theorem 4.3:** *Let $Q_1$ and $Q_2$ be two conjunctive queries over an $\mathcal{ALCNR}$ terminology $\mathcal{T}$, with the same number of distinguished variables. The problem of determining whether $Q_1$ is contained in $Q_2$ is decidable.* $\Box$

**Using existential entailment for recursive CARIN-$\mathcal{ALCNR}$:** As noted by Bürckert,[7] the existential entailment algorithm can be combined with constrained SLD-resolution [12] to provide a goal directed backward chaining algorithm on arbitrary (even recursive) CARIN-$\mathcal{ALCNR}$ Horn rules. This procedure will yield a refutation complete procedure for CARIN-$\mathcal{ALCNR}$ knowledge bases. That is, given a knowledge base $\Delta$ and a query $p(\bar{a})$, the algorithm will terminate if $\Delta \cup \neg p(\bar{a})$ is not satisfiable, but may not terminate otherwise. In the next sections we consider the problem of obtaining a *complete* reasoning algorithm for recursive CARIN knowledge bases.

# 5 Recursive CARIN-$\mathcal{ALCNR}$

In the previous section we showed that the reasoning problem is decidable for non recursive CARIN knowledge bases. We now consider what happens when the Horn rules are recursive. Recall that the reasoning problem for recursive function-free Horn rules without a terminology (i.e., datalog) is decidable [35] (and is even polynomial in the number of ground facts in the knowledge base).

We first show that the reasoning problem is undecidable for recursive CARIN-$\mathcal{ALCNR}$ knowledge bases. In fact, we show that the reasoning problem becomes undecidable simply by introducing either the constructor $\forall R.C$ or the constructor $(\leq\ n\, R)$. This result is interesting because these two constructors are generally considered to be at the core of most description logics.

In the next section we show that without these constructors we obtain a sublanguage of CARIN-$\mathcal{ALCNR}$ (called CARIN-MARC) for which the reasoning problem is decidable even when the Horn rules are recursive, as long as the terminology contains only concept definitions and they are acyclic. In Section 7 we describe another way of restricting the Horn rules (without restricting the description logic) such that the reasoning problem remains decidable.

The following theorem shows that if the description logic contains either the constructor $\forall\, R.C$ or $(\leq\ n\, R)$, then the reasoning problem is undecidable.

[7] personal communication.

**Theorem 5.1:** *The problem of determining whether $\Delta \models p(\bar{a})$ is undecidable, when $\Delta$ is a* CARIN-$\mathcal{L}$ *knowledge base with recursive function-free Horn rules, $\Delta$ has an acyclic terminological component that contains only concept definitions, and $\mathcal{L}$ is either*

1. *the description logic that includes only the constructor $\forall R.C$, or*

2. *the description logic that includes only the constructor $(\leq n\, R)$.*

□

The following theorem shows that introducing arbitrary (possibly cyclic) inclusion statements also causes the reasoning problem to be undecidable.

**Theorem 5.2:** *The problem of determining whether $\Delta \models p(\bar{a})$ is undecidable, when $\Delta$ is a* CARIN-$\mathcal{L}$ *knowledge base with recursive function-free Horn rules, the terminological component of $\Delta$ allows arbitrary inclusion statements and $\mathcal{L}$ includes either only the constructor $\exists R.C$ or only the constructor $(\geq n\, R)$.* □

The proofs of both theorems, given in the appendix, are obtained by encoding the execution of a Turing machine as a knowledge base of the form allowed in the theorems. Hence, we obtain a reduction from the halting problem to our decision problem.

# 6 Decidable Subset of Recursive CARIN-$\mathcal{ALCNR}$

We now show that in the language resulting from removing the constructors $\forall R.C$ and $(\leq n\, R)$ and not allowing terminological cycles the reasoning problem *is* decidable. Specifically, we consider the language CARIN-MARC that includes the constructors $\sqcap$, $\sqcup$, $(\geq n\, R)$, $\exists R.C$ and negation on primitive concepts.[8] Furthermore, CARIN-MARC allows only concept definitions in the terminological component (i.e., no inclusions or role definitions), and they must be acyclic. In what follows we describe a sound and complete inference procedure for CARIN-MARC. Our algorithm proceeds in two steps:

1. We first apply a set of propagation rules to an initial constraint system obtained from the knowledge base. The propagation rules we use are a variation on those used in Section 3. As before, the union of the completions will be equivalent to the original knowledge base.

2. Next, we evaluate the (possibly recursive) Horn rules in every completion. We show that a fact $p(\bar{a})$ is entailed by the knowledge base if and only if it is entailed in each of the completions that we construct. In the evaluation of the Horn rules we use a special procedure to check entailment of a ground atom of a concept or role predicate.

[8]Note that allowing arbitrary negation would allow us to express the constructors $\forall R.C$ and $(\leq n\, R)$.

## 6.1 The Inference Algorithm

### 6.1.1 Building the Initial Constraint System

Throughout the algorithm, we assume that all the concept definitions in $\Delta$ are unfolded. Given a knowledge base $\Delta$ whose terminology is $\mathcal{T}$, the algorithm begins by constructing an initial constraint system $S_\Delta$ as follows. If $C(a)$ is a ground fact in $\Delta$, where $C$ is defined in $\Delta_\mathcal{T}$ by $C := D$, we add $a : D$ to $S_\Delta$ (if $C$ does not have a definition in $\Delta_\mathcal{T}$, then we simply add $a : C$ to $S_\Delta$). If $R(a, b) \in \Delta$, then we add $aRb$ to $S_\Delta$. Finally, for every pair of constants $(a, b)$ in $\Delta$ we add the constraint $a \not\doteq b$ to $S_\Delta$.

### 6.1.2 Propagation Phase

The propagation rules we apply are shown in Figure 7. Rules 1 and 2 are the same as those shown in Section 3. Rule 3 is similar to rule 4 in Section 3, except that it does not necessarily create a new variable in the constraint system. It non-deterministically chooses either one of the existing successors of $s$, or adds a new successor. Rule 4, which is a variant of rule 5 in Section 3, adds to $s$ only the *minimal* number of $R$-successors needed in order to satisfy the $\geq$ constraint (as opposed to the rule in Section 3 that adds $n$ $R$-successors even when $s$ already has $R$-successors). Rule 5 is the choose rule that enforces every object to be an instance of a primitive concept or of its negation.

A constraint system is said to have a clash if it contains both $s : A$ and $s : \neg A$. As before, a constraint system is considered to be a *completion* when no propagation rule can be applied to it. We apply the rules using the same strategy as before.

**Remark 6.1:** One may ask at this point why we needed to design a new set of rules rather than simply taking the subset of the rules used in Section 3 for the constructors we kept in CARIN-MARC. The reason is that in Section 3, when rules 4 and 5 (i.e., $\rightarrow_\geq$ and $\rightarrow_\exists$) create *too many* successors for an object, then the application of the rule $\rightarrow_\leq$ would ensure (by equating some of the successors) that there is no clash-free completion in which the $\leq$ number-restriction is violated. However, since we do not have the $\rightarrow_\leq$ rule (since CARIN-MARC does not have the $\leq$ constructor), we need to modify the generating rules to ensure that only the minimal number of new objects is created.

It should be noted that rules 3 and 4 of this section *could* be used in Section 3. However, since rule 3 is both non deterministic and a generating rule, it will often lead to a larger number of completions.

Finally, another important property of the propagation rules in this section is the following. When a successor variable $v$ is generated in a constraint system, it is guaranteed that the size of the constraints on $v$ are smaller than the size of the constraints on the predecessor. Therefore, the application of the propagation rules is guaranteed to terminate without the need for an explicit termination condition. As a result, the construction of the canonical interpretations will also be simpler (we will not need any implicit links, because we don't have blocked variables). $\Box$

1. $S \rightarrow_{\sqcap} \{s : C_1, s : C_2\} \cup S$
   if 1. $s : C_1 \sqcap C_2$ is in $S$,
   2. $s : C_1$ and $s : C_2$ are not both in $S$.
2. $S \rightarrow_{\sqcup} \{s : D\} \cup S$
   if 1. $s : C_1 \sqcup C_2$ is in $S$,
   2. neither $s : C_1$ nor $s : C_2$ are in $S$,
   3. $D = C_1$ **or** $D = C_2$.
3. $S \rightarrow_{\exists} \{sRy, y : C\} \cup \{y \neq x \mid x \in s_R\} \cup S$
   1. $s : \exists R.C$ is in $S$,
   2. there is no $t$ such that $t$ is an $R$-successor of $s$ in $S$ and $t : C$ is in $S$,
   3. $y$ is a new variable **or** one of the existing $R$-successors of $s$ in $S$.
   4. $s_R$ is $Succ(s, R) \setminus y$.
4. $S \rightarrow_{\geq} \{sRy_1, \ldots, sRy_{n_0}\} \cup \{y_i \neq y_j \mid 1 \leq i, j \leq n_0, i \neq j\} \cup \{y_i \neq x \mid x \in Succ(s, R), 1 \leq i \leq n_0\} \cup S$
   if 1. $s : (\geq n\, R)$ is in $S$,
   2. $s$ has exactly $m$ $R$-successors in $S$, and $n = m + n_0$,
   3. $y_1, \ldots, y_{n_0}$ are new variables,
   4. there is no $l > n$, such that $s : (\geq l\, R)$ is in $S$.
5. $S \rightarrow_{\neg} \{s : D\} \cup S$
   if 1. $A$ is a primitive concept and both $s : A$ and $s : \neg A$ are not in $S$,
   2. $D = A$ **or** $D = \neg A$.

Figure 7: Propagation rules for recursive CARIN-MARC. $Succ(s, R)$ denotes the set of $R$-successors of $s$.

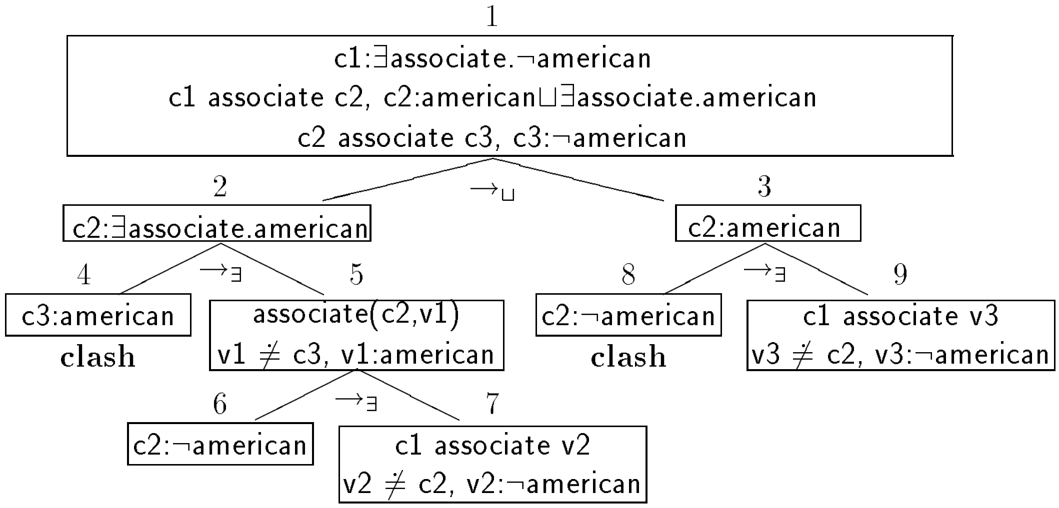


Figure 8: The application of propagation rules on $S_{\Delta_1}$. Note that in every node we show only the facts that were *added* to the constraint system. Under every node we show which propagation rule was applied to obtain its children.

**Example 6.1:** Consider the knowledge base $\Delta_1$ containing the following terminology:

american-associate := $\exists$associate.american
foreign-associate := $\exists$associate.$\neg$american
allied-company := american $\sqcup$ american-associate
conglomerate := ($\geq$ 2 associate)

the ground facts:

foreign-associate(c1), associate(c1,c2), allied-company(c2), associate(c2,c3), $\neg$american(c3).

and the rules:

$r_1$ : associate(X,Y) $\Rightarrow$ sameGroup(X,Y)
$r_2$ : sameGroup(X,Z) $\wedge$ sameGroup(Z,Y) $\Rightarrow$ sameGroup(X,Y)
$r_3$ : foreign-associate(X) $\wedge$ conglomerate(X) $\wedge$ sameGroup(X,Y) $\Rightarrow$ TaxLaw(Y,USA,Domestic)
$r_4$ : american-associate(X) $\wedge$ associate(X,Y) $\Rightarrow$ TaxLaw(Y,USA,Domestic).

The initial constraint system $S_{\Delta_1}$ includes:

c1 associate c2, c2 associate c3, c3:$\neg$american, c2: american$\sqcup$ $\exists$associate.american, and c1:$\exists$associate.$\neg$american.

Figure 8 shows the application of the propagation rules to the initial constraint system. We apply the $\rightarrow_\sqcup$ rule to c2, resulting in two possible constraint systems: node 2 (in which c2:$\exists$ associate.american is added), and node 3 (in which c2:american is added). In node 2 we apply the rule $\rightarrow_\exists$ to c2. The constraint c2:$\exists$ associate.american implies that c2 has at least one filler on the role associate that is American. There are two options. This filler may be an existing one, i.e., c3, as in node 4, however, this causes a contradiction with an existing constraint c3:$\neg$american. The second option is that there is another filler, v1, as in node 5. Since node 4 is contradictory, we do not consider it further. In node 5 we apply the rule $\rightarrow_\exists$ to c1. The constraint c1:$\exists$associate.$\neg$american implies that c1 has at least one filler on the role associate that is not American. Once again, there are two options, resulting in nodes 6 and 7. Similarly, we expand node 3 by applying the $\rightarrow_\exists$ to c1. $\Box$

### 6.1.3 Horn Rule Evaluation Step

In the second step of the algorithm we create a set of ground facts for every completion, and evaluate the Horn rule using a procedure described below.

Given a clash-free completion $S$, we create a set of ground facts $\Delta_S$ in a straightforward way as follows. The set of predicate names in $\Delta_S$ includes all the descriptions appearing in $S$ and the set of ordinary predicates and roles in $\Delta$. First, $\Delta_S$ contains all of the ground facts in $\Delta$. Second, we add to $\Delta_S$ facts corresponding to the constraints in $S$. Specifically, if $v : D \in S$, where $D$ is a description, then we add $D(v)$ to $\Delta_S$. If $sRt \in S$, we add $R(s,t)$ to $\Delta_S$ and if $v \neq u \in S$ we add $v \neq u$ to $\Delta_S$.

Given a completion $S$ and a query $q(\bar{a})$, we determine whether $\Delta_S \models q(\bar{a})$ using the conditions stated in the theorem below. Essentially, the theorem specifies how to entail a ground atom of a concept or role predicate. Entailment of ground atoms of ordinary predicates is done in the same fashion as in standard Horn reasoning algorithms. Our algorithm will return that the query $q(\bar{a})$ is entailed from $\Delta$ if and only if it is entailed from $\Delta_S$ for each clash-free completion $S$ of $S_\Delta$. The proof of the theorem is given in the appendix.

**Theorem 6.2:** *Let $S$ be a clash-free completion resulting from applying the propagation rules on $S_\Delta$. Let $\Delta_S$ be the the set of ground facts constructed for $S$.*

- *If $C(s)$ is an atom, where $C$ is a concept name defined in $\Delta_T$ by the description $D$, $\Delta_S \models C(s)$ if and only if:*
  - *$D$ is primitive or a negation of a primitive concept, and $D(s) \in \Delta_S$, or*
  - *$D = (\geq n\, R)$, and $s$ has at least $n$ $R$-successors in $S$, or*
  - *$D = \exists\, R.C$, and $s$ has an $R$-successor $t$ such that $\Delta_S \models C(t)$, or*
  - *$D = C_1 \sqcap C_2$, and $\Delta_S \models C_1(s)$ and $\Delta_S \models C_2(s)$, or*
  - *$D = C_1 \sqcup C_2$, and $\Delta_S \models C_1(s)$ or $\Delta_S \models C_2(s)$.*
- *If $R$ is a role, then $\Delta_S \models R(s,t)$, if and only if $R(s,t) \in \Delta_S$,*
- *If $p$ is an ordinary predicate, $\Delta_S \models p(\bar{a})$, if and only if $p(\bar{a}) \in S$ or, there exists a Horn rule $r \in \Delta$ of the form $p_1(\bar{X}_1) \wedge \ldots \wedge p_n(\bar{X}_n) \Rightarrow p(\bar{Y})$ and a mapping $\psi$ from the variables of $r$ to constants, such that $\psi(\bar{Y}) = \bar{a}$, and $\Delta_S \models \psi(p_i(\bar{X}_i))$, for $i$, $1 \leq i \leq n$.* $\Box$

**Example 6.2:** We illustrate the phase of evaluation of the Horn rules on two completions shown in Figure 8. Consider the completion described in node 9. The set of ground facts constructed for it, $\Delta_9$, contains the following facts that are originally in $S_{\Delta_1}$:

(∃associate.¬american)(c1), (american ⊔ ∃associate.american)(c2),
associate(c1, c2), associate(c2 c3), ¬american(c3),

and the following facts that correspond to constraints added during the propagation phase:

american(c2), associate(c1, v3), ¬american(v3), v3 ≠ c2 .

The facts sameGroup(c1,c2) and sameGroup(c2,c3) are entailed by $r_1$, and therefore rule $r_2$ entails sameGroup(c1,c3). Since company c1 has two fillers on the role associate (c2 and v3), it is an instance of conglomerate. It is also given that c1 is an instance of foreign-associate, and therefore, rule $r_3$ entails TaxLaw(c3,$USA$,$Domestic$).

Consider the completion in node 6 that has the following facts in addition to those from the initial database:

$(\exists$associate.american$)($c2$)$, associate(c2,v1), american(v1), $\neg$american(c2), v1 $\neq$ c3.

In this completion company c2 is an instance of american-associate, therefore, rule $r_4$ entails TaxLaw(c3,$USA$,$Domestic$). In fact, TaxLaw(c3, $USA$, $Domestic$) is entailed in all the clash-free completions, and therefore, it is entailed by $\Delta_1$. $\square$

## 6.2 Proof of Correctness and Complexity

In addition to Theorem 6.2, the key to proving the correctness of the algorithm is the following lemma, which is an analog of Lemma 3.4. The proof of the lemma is given in the appendix. We denote by $\Delta_g$ the set of ground facts of ordinary predicates in $\Delta$.

**Lemma 6.3:** *Let $S$ be a clash-free constraint system generated by applying a (possibly empty) sequence of propagation rules to $S_\Delta$. Let $S_1, \ldots, S_l$ be the constraint systems that can be generated from $S$ by applying one of the propagation rules. Let $q(\bar{a})$ be a ground atom. Then, $S \cup \Delta_r \models q(\bar{a})$ if and only if $S_i \cup \Delta_r \cup \Delta_g \models q(\bar{a})$ for every $i$, $1 \leq i \leq l$.* $\square$

The soundness and completeness of our algorithm is established by the following theorem:

**Theorem 6.4:** *Let $\Delta$ be a* CARIN-MARC *knowledge base. $\Delta \models q(\bar{a})$ if and only if $\Delta_S \cup \Delta_r \cup \Delta_g \models q(\bar{a})$ for every $S$ that is a clash-free completion of $S_\Delta$.* $\square$

**Proof:** Since we unfolded the concept definitions when creating $S_\Delta$ it follows that $M(\Delta) = M(S_\Delta \cup \Delta_r \cup \Delta_g)$. By induction on the application of the propagation rules in the first phase of the algorithm, Lemma 6.3 implies that $\Delta \models q(\bar{a})$ if and only if $S \cup \Delta_r \cup \Delta_g \models q(\bar{a})$ for every clash-free completion $S$ of $S_\Delta$. Since $\Delta_S$ is equivalent to $S \cup \Delta_g$, Theorem 6.2 entails that $\Delta \models q(\bar{a})$ if and only if $\Delta_S \cup \Delta_r \models q(\bar{a})$ for every clash free completion $S$ of $S_\Delta$. $\square$

### 6.2.1 Complexity

The complexity of reasoning in CARIN-MARC is given by the following theorem:

**Theorem 6.5:** *Let $\Delta$ be a* CARIN-MARC *knowledge base. Deciding whether $\Delta \models q(\bar{a})$ is co-NP-Complete in the number of ground facts in $\Delta$, and polynomial in the number of Horn rules in $\Delta$. If the numbers in the number restrictions in $\Delta$ are encoded in unary form, the entailment problem is CO-NP-Complete in the size of the terminology of $\Delta$.* $\square$

**Proof:** We begin with the complexity in the number of Horn rules in $\Delta$. The size and the number of completions is independent of the number of Horn rules. In each completion we can compute the least fixed point model by a bottom-up evaluation of the Horn rules, which is polynomial in the number of rules.

Consider the number of ground facts in $\Delta$. The number of times a propagation rule can be applied to an object in a constraint system is polynomial in the size of the terminology, and does not depend on the number of ground facts in $\Delta$. Each application of a propagation rule adds a constant number of constraints. Furthermore, if $s$ is a successor of an individual $a$ in a constraint system $S$, then the distance of $s$ from $a$ is bounded by the size of the largest concept in the terminology. Therefore, the number of objects in a completion is linear in the number of ground facts in $\Delta$, and hence every completion is obtained by a linear number of applications of propagation rules. This entails that the size of each completion is linear in the number of ground facts. To show that a query $q(\bar{a})$ is not entailed from $\Delta$, we need to find one completion $S$ of $S_\beta$ in which $q(\bar{a})$ is not part of the least fixed point model of $\Delta_S \cup \Delta_r$. Computing the least fixed point model can be done in time polynomial in the size of the completion, and therefore, showing that $\Delta \not\models q(\bar{a})$ is in NP.

To prove the NP-hardness result we use a reduction from the NP-complete problem 3-SAT. We encode a 3-SAT propositional theory $\Delta_p$ by a CARIN-MARC knowledge base $\Delta$ using a relation

$$R(clauseNumber, positionInClause, signOfLiteral, variableNumber).$$

For example, if $\{p_1, \neg p_2, p_4\}$ is the first clause in $\Delta_p$, then $\Delta$ will include the ground facts $R(1, 1, Plus, 1)$, $R(1, 2, Minus, 2)$ and $R(1, 3, Plus, 4)$.

A truth value assignment for a propositional theory is given by the concept $A$. The atom $A(v)$ denotes that variable $v$ is assigned $True$.

In addition to the ground facts for $R$, $\Delta$ contains 8 rules (corresponding to the 8 different forms of a clause) that define a predicate $NS$. The ground atom $NS(n)$ denotes that clause $n$ is not satisfied under the current variable assignment. For example, the following rule considers clauses in which the first literal is positive and the second two are negative:

$$R(n, 1, Plus, v_1) \wedge R(n, 2, Minus, v_2) \wedge R(n, 3, Minus, v_3) \wedge \neg A(v_1) \wedge A(v_2) \wedge A(v_3) \Rightarrow NS(n).$$

Finally, we have the rule

$$NS(n) \Rightarrow NSAT.$$

$NSAT$ is entailed only when one of the clauses is not satisfied. It is easy to see that if there is some satisfying assignment to the variables of $\Delta_p$, we can build an extension for the concept $A$ that includes exactly the propositions that are mapped to $True$. From this extension of $A$, we can build a model of $\Delta$ in which $NSAT$ is not satisfied. On the other hand, if any assignment to the variables of $\Delta_p$ always causes one of the clauses not to be satisfied, then $\Delta \models NSAT$. Note that the size of $\Delta$ is linear in the size of $\Delta_p$.

Finally, consider the size of the terminology. The hardness result follows from the complexity of concept unsatisfiability in $\mathcal{ALU}$ [33]. Given a concept $C$, the entailment $C(a) \models False$ holds if and only if $C$ is not satisfiable. As in our analysis above, if the numbers in the number restrictions are encoded in unary form, then the size of each completion is polynomial in the size of the terminology. Checking entailment of ground atom in a completion can also be done in time polynomial in the size of the terminology. Hence, the bottom-up evaluation of the rules can be done in time polynomial in the size of the terminology. Therefore, non-entailment is in NP. $\Box$

# 7 CARIN-$\mathcal{ALCNR}$ with Role-Safe Rules

In this section we describe another way of obtaining a subset of CARIN-$\mathcal{ALCNR}$ for which sound and complete inference is possible for recursive function-free rules, while still allowing *all* the constructors of $\mathcal{ALCNR}$ and *arbitrary* inclusion statements in the terminology. The subset language is obtained by restricting the Horn rules in the knowledge base to be *role-safe*, as we define below. Role-safe rules restrict the way in which variables can appear in role atoms in the rules. This restriction is similar in spirit to the *safety* condition which is widely employed in database query languages for queries containing interpreted predicates (e.g., $\leq$, $<$ $\neq$) [35]. Furthermore, many classical uses of recursion (e.g., connectivity on graphs whose edges are represented by ordinary predicates) can be expressed by role-safe rules. An ordinary predicate $e$ is said to be a *base* predicate in $\Delta$ if $e$ does not appear in the consequent of any Horn rule in $\Delta$.

**Definition 7.1:** *A rule $r$ is said to be role-safe if for every atom of the form $R(x, y)$ in the antecedent, where $R$ is a role, then either $x$ or $y$ appear in an atom of a base predicate in the antecedent of $r$.* $\Box$

The following theorem shows that the reasoning problem in CARIN-$\mathcal{ALCNR}$ is decidable when all the Horn rules in the KB are role-safe.

**Theorem 7.1:** *Let $\Delta$ be a CARIN-$\mathcal{ALCNR}$ knowledge base in which all Horn rules are role-safe. The problem of determining whether $\Delta \models q(\bar{a})$ is decidable.* $\Box$

It should be noted that CARIN-$\mathcal{ALCNR}$ with role-safe rules is a strictly more expressive language than AL-Log [14], since AL-Log only allows concept atoms in the Horn rules. The complexity of reasoning with role-safe rules is co-NP-Complete in the number of ground facts in $\Delta$, polynomial in the number of Horn rules in $\Delta$, and doubly exponential in the size of the terminology of $\Delta$.

**Proof:** The inference algorithm is exactly the one we used in Section 4 for non recursive CARIN-$\mathcal{ALCNR}$ knowledge bases, except that we use 0-tree equivalence as the termination condition (i.e., the same condition as in [11]), and in the Horn-rule evaluation phase, the rules may be recursive.

The key to the proof of soundness is to note that in the bottom-up evaluation of the Horn rules we do not make use of the implicit tuples in $I_S$, but only of explicit tuples. To see this, consider a mapping $\psi$ from the variables of a rule $r \in \Delta_r$ to objects in $\mathcal{O}^{I_S}$. If $R(x, y)$ is an atom in the antecedent of $r$, then either $x$ or $y$ appears in an atom of a base predicate in the antecedent of $r$, and therefore, either $\psi(x)$ or $\psi(y)$ is an individual in $S$. Since there are no arcs from individuals to blocked variables, then both $\psi(x)$ and $\psi(y)$ are not blocked variables, and therefore $(\psi(x), \psi(y))$ is an explicit tuple in $I_S$.

Since the explicit tuples must exist in *every* model of $S$, the facts we infer for the ordinary predicates are entailed by $S$. As for completeness, if we have a clash-free completion from which we could not derive $q(\bar{a})$, then it provides an example model of $\Delta$ in which $q(\bar{a})$ is not entailed. $\Box$

**Example 7.1:** We illustrate the algorithm with the following simple example. Consider a knowledge base $\Delta_2$ that contains the concept $C$, the role $R$, and the ordinary binary predicates $e$ and $p$. The terminology has the single cyclic inclusion statement $C \sqsubseteq \exists R.C$, and we have the ground facts $C(a), C(b), e(a,b)$ and $e(b,c)$. Finally, we have the rules:

$s_1 : e(x,y) \wedge R(x,z) \Rightarrow p(x,y)$
$s_2 : p(x,z) \wedge p(z,y) \Rightarrow p(x,y)$.

The propagation step would create the completion that includes the following constraints in addition to those in the initial constraint system: $aRv_1$, $v_1 : C$, $v_1Rv_2$, $v_2 : C$, $bRu_1$, $u_1 : C$, $u_1Ru_2$ and $u_2 : C$, where $v_1$, $v_2$, $u_1$ and $u_2$ are newly created variables in the constraint system. We create a model $I$ of the completion as follows. The domain of $I$ is $\{a, b, c, v_1, v_2, u_1, u_2\}$. The interpretations of the concepts and roles are

$C^I = \{a, b, v_1, v_2, u_1, u_2\}$,
$R^I = \{(a, v_1), (v_1, v_2), (v_2, v_2), (b, u_1), (u_1, u_2), (u_2, u_2)\}$.

The interpretation of $e$ is taken directly from the ground facts in $\Delta_2$: $e^I = \{(a,b), (b,c)\}$. Finally, the interpretation of $p$ is constructed from the rules in $\Delta_2$: $p^I = \{(a,b), (b,c), (a,c)\}$.

Therefore, $\Delta_2$ entails $p(a,b)$, $p(b,c)$ and $p(a,c)$. The important point is to note that the extension of $R$ includes the tuples $(v_2, v_2)$ and $(u_2, u_2)$ that are not explicit in the completion, but are necessary in order to obtain a finite model, while satisfying the inclusion $C \sqsubseteq \exists R.C$. However, because of the fact that the rules are role-safe, these tuples are not used in computing the extension of $p$. $\Box$

# 8 Conclusions

We described CARIN, a family of representation languages that combine the expressive power of Horn rules and description logics. We addressed the issue of designing sound and complete inference procedures for CARIN knowledge bases. We identified the core inference problem of existential entailment, and showed that it is central to several reasoning problems in CARIN. We described an existential entailment algorithm for $\mathcal{ALCNR}$. As a result, we obtained a sound and complete algorithm for reasoning in non recursive CARIN-$\mathcal{ALCNR}$ knowledge bases, and an algorithm for query containment over $\mathcal{ALCNR}$. We have shown that in general, the reasoning problem for recursive CARIN-$\mathcal{ALCNR}$ knowledge bases is undecidable, and identified the constructors of $\mathcal{ALCNR}$ causing the undecidability. Finally, we have shown two ways in which recursive CARIN-$\mathcal{ALCNR}$ knowledge bases can be restricted while obtaining sound and complete reasoning.

CARIN has already proved useful in two contexts. In [22] it is shown how the expressive power of CARIN has been key to the development of the Information Manifold system that combines information from multiple autonomous and heterogeneous data sources. In particular, the ability to combine relations of arbitrary arity (which are needed when modeling relational databases) with a hierarchy of concepts expressed in a description logic terminology has proved very useful in that application. In contrast, related systems (e.g., SIMS [3], Razor [16]) use only description logics or only Horn rules. Furthermore, the ability to answer conjunctive queries over a description logic knowledge base, and to decide query containment were a key in developing the architecture of the system. Deciding containment of conjunctive queries is the key building block in determining which data sources are relevant to a user query [22].

Another use of CARIN for the problem of knowledge base verification is described in [25]. In that paper, the knowledge base verification problem is shown to be related to the *query containment* problem studied in the database literature [35]. Our query containment algorithm extends containment algorithms to conjunctive queries over description logic knowledge bases. Consequently, this algorithm enables verifying hybrid knowledge bases containing both Horn rules and description logics. Furthermore, we have shown in [25] that our query containment algorithm also enables us to deal with tuple-generating dependencies. Tuple generating dependencies (tgd's) are logical sentences of the form

$$\exists \bar{X} p_1(\bar{X}_1) \wedge \ldots \wedge p_n(\bar{X}_n) \Rightarrow \exists \bar{Y} q_1(\bar{Y}_1) \wedge \ldots \wedge q_m(\bar{Y}_m).$$

Tgd's are useful in expressing integrity constraints on rule-based knowledge bases. Verifying the correctness of a set of rules requires reasoning about entailment among tgd's (i.e., deciding whether one tgd entails another). The entailment problem for tgd's is known to be undecidable in general [36, 18]. In [25] we show that in some cases, entailment between tgd's can be translated into query containment of conjunctive CARIN queries. As a result, we obtain new decidability results for the entailment problem for tgds.

## Related Work

Several other works have discussed the integration of Horn rules and description logics. Some works (e.g., AL-log [14], TaxLog [1], LIFE [2, 30]) had the goal of using a description logic or other object-oriented component as a *typing* language on the variables already appearing in the rules (which could also be recursive). For example, in AL-log [14], which is most closely related to CARIN, only unary predicates from the description logic are allowed in the Horn rules, and the variables used in atoms of concepts must appear in atoms of ordinary predicates as well. AL-log allows recursive Horn rules, but a weaker description logic, $\mathcal{ALC}$, and it is shown in [14] that the reasoning problem is decidable in the language. Other works (e.g., [9, 19]) considered a more tight integration of the two formalisms. For example, KRYPTON [9] combined an assertional component (more expressive than Horn rules) with a less expressive description logic than $\mathcal{ALCNR}$. The reasoning engine was modified by either adding resolution steps to consider the inferences sanctioned by the terminological component, or by modifying the unification operation underlying the resolution engine. These approaches are either incomplete or guarantee only refutation completeness.

LIFE [2] is also a language whose goal is to combine logic programming with a structure oriented component. However, the LIFE structure-oriented component is composed of $\psi$-*terms* that differ from description logics in several significant ways. The idea of $\psi$-*terms,* rooted in the functional programming paradigm, is to represent subtyping in record-like data structures. On one hand, they are more limited than description logics and closer to feature logics [34], since they only allow attributes, i.e functional roles (as opposed to roles with multiple fillers). For example, number restrictions and existential statements about role fillers that are standard in description logics are not expressible in $\psi$-*terms*. On the other hand, the variables in $\psi$-*terms* enable to express complex coreferences constraints, which can only be expressed in a limited fashion using the same-as constructor in description logics.[9]

A different approach to integrating rules and description logics is to add rules as an additional constructor (e.g., CLASSIC [8], BACK [31], LOOM [28]). These works allowed only rules of a restricted form: $C(x) \Rightarrow D(x)$, where C and D are concepts. Furthermore, the rules are generally not integrated in subsumption inferences but they are just used to derive additional knowledge about concept instances. MacGregor [29] and Yen [39] describe algorithms for determining rule-specificity and classification of arbitrary predicates in LOOM, which are an instance of the existential entailment problem described here. However, since subsumption in LOOM is undecidable, their algorithms are not complete either.

Our analysis of CARIN focussed on the time complexity of the reasoning problem. We showed that for CARIN-MARC, the complexity is co-NP Complete in the number of ground facts. The question arises whether there exists subset of CARIN that are able to express *all* queries in coNP. Recently, Cadoli et al. [13] investigated the expressive power of CARIN, and showed that there are certain classes of second order formulas, such that even a relatively simple subset of CARIN, role-safe CARIN-MARC is able to express all queries in those classes.

## Future Work

It is important to emphasize that the focus of this paper has been on the question of decidability of the reasoning problem in CARIN-$\mathcal{ALCNR}$. Our work raises the important issue of how to *efficiently* reason in systems based on CARIN. One direction to investigate is to find subsets of CARIN-$\mathcal{ALCNR}$ for which the resulting language is more tractable than CARIN-$\mathcal{ALCNR}$. A second direction is to find practically efficient methods for implementing reasoning in CARIN.

One of the possible optimizations we plan to consider is to reduce the size and number of the completions created by the algorithm by employing a termination condition in the spirit of the one proposed in [4]. In that work, Baader et al. use a termination condition that modifies the one used by [11], by not requiring a blocked variable to have the same value for the $\sigma$ function as its witness, but rather have a *subset* of the $\sigma$ value of its witness. Clearly, employing this more relaxed condition reduces the number of objects in a completion. In our context, we need to extend the condition of [4] to $n$-tree equivalence.

We have already found two applications of CARIN in information integration and in verification of knowledge bases. We are currently looking into applying CARIN as a representational tool for modeling physical devices, for describing ontologies, and for database applications such as datawarehousing and schema integration.

[9] One main reason for the limited coreference constraints in description logics is that subsumption becomes undecidable when coreference constraints are applied to roles with multiple fillers.

# 9 Appendix

**Proof of Lemma 3.4:** We prove the claim for each of the propagation rules. We begin with the non generating and deterministic rules, for which the set $\mathcal{S}$ is a singleton set containing the constraint system $S'$. We show that $I$ is a model of $S$ if and only if $I$ is a model of $S'$, and therefore the lemma holds for these rules.

Trivially, every model of $S'$ is a model of $S$ because $S$ contains a subset of the constraints of $S'$. For the other direction, let $I$ be a model of $S$. There are several cases:

- Rule 1: If $I$ is a model of $S$ and $s : C_1 \sqcap C_2 \in S$, then $\alpha^{\mathcal{I}}(s) \in [C_1 \sqcap C_2]^I$, therefore, by definition, $\alpha^{\mathcal{I}}(s) \in C_1^I$ and $\alpha^{\mathcal{I}}(s) \in C_2^I$. Since $S' = S \cup \{s : C_1, s : C_2\}$, then $I$ is a model of $S'$.

- Rule 3: Since the propagation rule is applied, and $I$ is a model of $S$, there exist $s$ and $t$ such that $(\alpha^{\mathcal{I}}(s),\alpha^{\mathcal{I}}(t)) \in R^I$. Since $s : \forall R.C \in S$, by the definition of the extension of $\forall R.C$, $\alpha^{\mathcal{I}}(t) \in C^I$, and therefore, since $S' = S \cup \{t : C\}$, $I$ is a model of $S'$.

- Rule 7: In this case, $S' = S \cup s : C$. However, $\forall x.x : C \in S$, therefore, $I$ is a model of $S'$.

We now consider the other propagation rules. Consider rule 2, and let $S_1$ and $S_2$ be the two constraint systems that can be obtained by applying the rule to $S$. The set of models of $S$ is the union of the models of $S_1$ and $S_2$, and therefore the claim of the lemma holds.

Consider rule 4, and let $S'$ be the single constraint system resulting from applying the rule to $S$. Denote by $y$ the variable that is added to the constraint system $S$ while applying the rule to the variable $s$. Note that any model $I'$ of $S'$ is obtained from a model $I$ of $S$ by extending the mapping $\alpha^I$ to the new variable $y$. Let $I$ be a model of $S$. For one direction of the lemma, it suffices to show that $I' \models Q$ for every model $I'$ of $S'$ that is obtained from $I$ by extending $\alpha^I$ to $y$. Since $I \models Q$, there is a mapping $\sigma$ from the variables and constants of $Q$ to $\mathcal{O}^I$ that maps every literal in one of the $Q_i$'s to a tuple in extensions of the relations in $I$. The same mapping $\sigma$ will be valid in $I'$ as well, because the set of tuples in the extensions is the same as in $I$. Therefore $I' \models Q$.

For the other direction, suppose that $S' \models Q$, and we show that $S \models Q$. Let $I$ be a model of $S$. Since $I$ is a model of $S$, there must be some object $o \in \mathcal{O}^I$, such that $(\alpha^I(s), o) \in R^I$, and $o \in C^I$. Consider the interpretation $I'$ obtained by extending $I$ by $\alpha^{I'}(y) = o$. Clearly, $I'$ is a model of $S'$. Therefore, $I \models Q$, because $I' \models Q$ and the models $I$ and $I'$ have identical relation extensions.

The proof for rule 5 is similar to that of rule 4. As before, every model of $S'$ is obtained from a model of $S$ by extending $\alpha^I$ to $y_1, \ldots, y_n$. If $I$ is a model of $S$ and $I'$ is a model of $S'$ that is obtained from $I$, then, in the same way shown for rule 4, if $I \models Q$ then $I' \models Q$. For the other direction, suppose $S' \models Q$, and we show that $S \models Q$. Let $I$ be a model of $S$, and therefore there are distinct objects $o_1, \ldots, o_n$ in $\mathcal{O}^I$, such that $(\alpha^I(s), o_i) \in R^I$, for $i$, $1 \le i \le n$. Consider the interpretation $I'$ obtained by extending $I$ by $\alpha^{I'}(y_i) = o_i$. $I'$ is a model of $S'$, and therefore, as before $I \models Q$.

Finally, consider rule 6, and suppose the constraint system $S'$ was obtained from $S$ by replacing $y$ by $t$. In this case, every model $I$ of $S$ can be obtained from a model $I'$ of $S'$ by extending $\alpha^{I'}$ to $y$. For the first direction, suppose $S' \models Q$ for every $S \in \mathcal{S}$, and let $I$ be a model of $S$. We need to show that $I \models Q$. Since $I$ is obtained from extending a model $I'$ of some $S \in \mathcal{S}$, the same variable mapping from the variables of $Q$ to $\mathcal{O}^{I'}$ that shows that $I' \models Q$ will show that $I \models Q$. For the second direction, suppose $S \models Q$, and let $I'$ be a model of $S' \in \mathcal{S}$. We need to show that $I' \models Q$. Let $I$ be the model of $S$ obtained by extending $\alpha^{I'}$ by setting $\alpha^{I}(y)$ to $\alpha^{I'}(t)$. It can be checked that $I$ is indeed a model of $S$, because any constraint involving $y$ in $S$ appears in $S'$ as constraints where $y$ is replaced by $t$. Therefore, the same variable mapping that shows $I \models Q$ will show that $I' \models Q$. $\Box$

The proof of Theorem 5.1 is easier to illustrate after the proof of Theorem 5.2.

**Proof of Theorem 5.2:** We begin with the case in which the description logic contains only the constructor $\exists R.C$. We reduce the halting problem of a Turing machine $TM$ to the entailment problem. We assume without loss of generality that $TM$ begins with the empty string on its tape. The initial state of $TM$ is $Q_0$ and its halting state is $Q_h$. An execution of a Turing machine can be described by a set of *configurations*, each describing the tape contents, head position and state of the machine at a given time point. We encode the configurations of $TM$ by a CARIN knowledge base $\Delta$. Configuration times and tape positions are represented by instances of concept $integer$ in our encoding. The role $succ(x, y)$ is intended to represent that $y$ is the successor integer to $x$. The knowledge base $\Delta$ includes the following statements about integers:

$integer(1)$,
$integer \sqsubseteq \exists succ.integer$.

The relation $lt(x, y)$ is intended to represent that $x$ is less than $y$, and is defined by the following recursive rules in $\Delta$:

$succ(x, y) \Rightarrow lt(x, y)$.
$succ(x, z) \wedge lt(z, y) \Rightarrow lt(x, y)$.

The relation $state(t, q)$ is intended to represent that the machine is in state $q$ at time $t$. The relation $headPos(t, p)$ is intended to represent that the machine's head is at position $p$ on the input tape at time $t$, and the relation $tape(t, p, s)$ is intended to represent that in time $t$, the tape has the symbol $s$ in position $p$. The following ground facts describe the initial state of the machine:

$state(1, Q_0)$, $headPos(1, 1)$,
$integer(t) \Rightarrow tape(1, t, \text{""})$.

Next we describe the rules corresponding to the transitions of the Turing machine $TM$. The rules for transitions differ slightly depending on whether the head is moved to the left or to the right, so below we describe the rules for the transition $\delta(Q, A) = (Q', A', \Rightarrow)$, i.e., when the machine is in state $Q$ and reading the symbol $A$, the machine writes the symbol $A'$ on the tape, moves one place to the right and goes into state $Q'$.

The rules need to describe: (1) the change of state, (2) change of head position, and (3) changes to the tape. The rule for change of state is the following:

$r_1 : integer(c) \wedge integer(c_1) \wedge succ(c, c_1) \wedge state(c, Q) \wedge headPos(c, p) \wedge tape(c, p, A) \Rightarrow state(c_1, Q')$.

The rule for changing the head position is:

$r_2 : integer(c) \wedge integer(c_1) \wedge succ(c, c_1) \wedge state(c, Q) \wedge headPos(c, p) \wedge succ(p, p_1) \wedge tape(c, p, A) \Rightarrow headPos(c_1, p_1)$.

The rule for changing the contents of the tape is:

$r_3 : integer(c) \wedge integer(c_1) \wedge succ(c, c_1) \wedge state(c, Q) \wedge headPos(c, p) \wedge tape(c, p, A) \Rightarrow tape(c_1, p, A')$.

The following rules are needed to state what *did not* change on the tape: (the first rule takes care of the symbols to the right of the head and the second takes care of those to its left).

$r_4 : integer(c) \wedge integer(c_1) \wedge succ(c, c_1) \wedge state(c, Q) \wedge headPos(c, p) \wedge tape(c, p, A) \wedge lt(p, y) \wedge tape(c, y, x) \Rightarrow tape(c_1, y, x)$.

$r_5 : integer(c) \wedge integer(c_1) \wedge succ(c, c_1) \wedge state(c, Q) \wedge headPos(c, p) \wedge tape(c, p, A) \wedge lt(y, p) \wedge tape(c, y, x) \Rightarrow tape(c_1, y, x)$.

Finally, the following rule defines a predicate *query*:

$r_6 : integer(c_1) \wedge state(c_1, Q_h) \wedge lt(1, c_1) \Rightarrow query$.

The proof of Theorem 5.2 follows from the following claim:

**Claim:** The machine $TM$ halts on the empty string if and only if $\Delta \models query$. $\Box$

**Proof:** We first define the *intended model*, $M$, of $\Delta$. The domain of $M$ includes the integers, the states of $TM$, and symbols in the alphabet of $TM$. The extension of *integer* includes exactly all the integers greater or equal to 1, and the extension of *succ* is $(i, i+1)$, for every $i \geq 1$. The extensions of *state*, *headPos* and *tape* are the minimal model of $\Delta$ that includes the extension of *integer* and *succ*, and in which $(1, Q_0) \in state^M$ and $(1, 1) \in headPos^M$. Note that this model is unique.

It is easy to prove by induction that $M$ describes exactly the execution of $TM$, i.e.,

- $TM$ is in state $q$ at time $i$ if and only if $(i, q) \in state^M$,
- The head of $TM$ is in position $p$ at time $i$ if and only if $(i, p) \in headPos^M$, and
- The tape contains the symbol $a$ in position $p$ at time $i$ if and only if $(i, p, a) \in tape^M$.

We begin with the *if* direction. If $\Delta \models query$, there is an integer $j$, for which the rule $r_6$ is satisfied. At time $j$ the machine will be in the halting state. Therefore, we have shown that if $\Delta \models query$ then $TM$ halts on the empty string.

Consider the *only-if* direction. Assume that the machine $TM$ halts, and let $M_1$ be a model of $\Delta$. We need to show that $M_1 \models query$. We define a mapping $\psi$ from the integers to the domain of $M_1$, $\mathcal{O}^{M_1}$. We define $\psi(1) = 1^{M_1}$. We define the mapping for the other integers inductively. Since $M_1$ is a model of $\Delta$ (and therefore of the inclusion statement), there exists at least one element $s$ in the domain of $M_1$, such that $(1^{M_1}, s) \in succ^{M_1}$. We choose one such $s$ arbitrarily and define $\psi(2) = s$. Similarly, we define the mapping for the integers greater than 2. Note that it is possible that $\psi(i) = \psi(j)$ for some $i \neq j$. The following claims follow by induction on $i$:

- If $TM$ is in state $Q$ at time $i$, then $(\psi(i), Q^{M_1}) \in state^{M_1}$,
- If the head of $TM$ is in position $p$ at time $i$, then $(\psi(i), \psi(p)) \in headPos^{M_1}$,
- If the tape contains the symbol $A$ in position $p$ at time $i$, then $(\psi(i), \psi(p), A^{M_1}) \in tape^{M_1}$.

This induction claim holds for $i = 1$ because of the facts in $\Delta$ that describe the initial state. The inductive step follows by examining the rule corresponding to the transition that $TM$ takes in time $i$, and noting that the induction hypothesis guarantees that the antecedent of the rule is satisfied. Note that $M_1$ does not necessarily encode the execution of the machine precisely, but *contains* the tuples that describe the execution. Consequently, since $TM$ halts after $N$ steps, *query* will follow by the substitution $x \rightarrow \psi(N)$ in rule $r_6$.

Consider the second case of the theorem, i.e., the case in which the description logic contains the constructor $(\geq n\, R)$. In this case, in our construction of $\Delta$ we replace the inclusion $integer \sqsubseteq \exists succ.integer$ by the inclusion $\top \sqsubseteq (\geq 1\, succ)$. In the proof, the new inclusion guarantees that we can construct the mapping $\psi$ for every integer (because every object has a successor). In order for the intended model to satisfy the inclusion $\top \sqsubseteq (\geq 1\, succ)$ for objects that are not instances of *integer*, we add to the model a new object $fin$, such that $(fin, fin)$ is in the extension of *succ*, and $(o, fin)$ is also in the extension of *succ* for every object $o$ which is not in the extension of *integer*. The rest of the proof is similar to the previous case. □

**Proof of Theorem 5.1:** Consider the case in which $\mathcal{L}$ has only the constructor $(\leq n\, R)$. We define $\Delta_1$ which is a slight modification of $\Delta$ of the previous proof as follows. The role predicate *integer* in $\Delta$ is now an ordinary predicate in $\Delta_1$, and $\Delta_1$ does not have any inclusion statements. Instead of the inclusion statements, $\Delta_1$ contains the following Horn rules:

$s_1 : integer(x) \wedge succ(x, y) \Rightarrow integer(y)$
$s_2 : integer(x) \wedge (\leq 0\, succ)(x) \Rightarrow query.$

The first direction of the proof is the same as before, using the intended model $M$ for $\Delta_1$. Note that the second atom in the antecedent of $s_2$ is not satisfied in $M$. For the other direction, consider a model $M_1$ of $\Delta_1$. There are two cases. If we can construct the mapping $\psi$ from the integers to $\mathcal{O}^{M_1}$ as before (i.e., there is an infinite sequence of integers), the same proof holds, and therefore, $M_1 \models query$. If not, then there is some integer $n$, such that there is no tuple of the form $(\psi(n), j)$ in $succ^{M_1}$ and $\psi(n) \in integer^{M_1}$. Therefore, since $M_1$ is a model of $s_2$, it must be the case that $M_1 \models query$.

Consider the case in which $\mathcal{L}$ contains only the constructor $\forall R.C$. We define $\Delta_2$ by modifying $\Delta_1$ as follows. Instead of the rule $s_2$ we add the following rule:

$s_3 : integer(x) \wedge (\forall\, succ.B)(x) \Rightarrow query.$

where $B$ is a new concept predicate. In the proof, the intended model $M$ of $\Delta_2$ will have the empty extension for $B$. The proof of the first direction follows as before, because the antecedent of $s_3$ is never satisfied in $M$. For the other direction, consider a model $M_1$ of $\Delta_2$. If $M_1$ is a model in which there is some element $o \in integer^{M_1}$ such that $o \in (\forall\, succ.B)^{M_1}$, then $M_1 \models query$ because of $s_3$. If not, then *every* element $o \in integer^{M_1}$ must have a successor, i.e., there must exist an $o_1$ such that $(o, o_1) \in succ^{M_1}$ (otherwise, $o \in (\forall\, succ.B)^{M_1}$). Hence, we can build the mapping $\psi$ as in the previous proofs, and show that $M_1 \models query$. $\Box$

**Proof of Theorem 6.2:** Let $S$ be a clash-free completion of $S_\Delta$ and let $\Delta_S$ be the set of ground facts constructed for $S$. Recall that $\Delta_r$ denotes the set of Horn rules in $\Delta$. We define a *canonical model* $M_S$ for $\Delta_S \cup \Delta_r$. The domain of $M_S$ includes all the constants in $\Delta_S$, and for each constant $s^{M_S} = s$. The extensions of the relations are defined as follows. If $A$ is a primitive concept, then $s \in A^{M_S}$ if and only if $A(s) \in \Delta_S$. For a role $R$, $(s,t) \in R^{M_S}$ if and only if $R(s,t) \in \Delta_S$. The extensions of the complex concepts are determined by the equations in Section 2.1. The extension of each of the ordinary predicates in $M_S$ is determined as follows. We begin with the ground facts in $S_\Delta$, i.e., if $e(\bar{a}) \in \Delta_S$, then $\bar{a} \in e^{M_S}$. Next, we compute the least fixed point model of the ordinary predicates that satisfies the Horn rules, and contains the extensions of the role and concept predicates as defined above. This model can be computed by a bottom-up evaluation of the Horn rules. By induction on the size of the descriptions appearing in concept atoms in $\Delta_S$, and because $S$ is a completion, it can be shown that $M_S$ is a model of $\Delta_S$.

We begin the proof with the case of concept atoms, and consider the different forms of concepts. In the proof, $M_S$ acts as a counterexample model for the *only-if* direction.

- Consider an atom of the form $A(s)$, where $A$ is a primitive concept. If $A(s) \in \Delta_S$, then clearly $\Delta_S \models A(s)$. If $A(s) \notin \Delta_S$, then $M_S$ is a model of $\Delta_S$ in which $A(s)$ is not satisfied, and hence $\Delta_S \not\models A(s)$. The same argument holds for an atom of the form $\neg A(s)$, where $A$ is a primitive concept.

- Consider an atom of the form $(\geq\ n\, R)(s)$. If $s$ has at least $n$ $R$-successors in $S$, then, $\Delta_S \models (\geq\ n\, R)(s)$. This follows because all $R$-successors of a given object in a constraint system are separated from each other (note that this property holds in the initial constraint system and is conserved by the application of rules 3 and 4). If $s$ does not have $n$ $R$-successors in $S$, then $M_S$ is again a counterexample model that shows that $\Delta_S \not\models (\geq\ n\, R)(s)$.

- Consider an atom of the form $(\exists R.C)(s)$. If $s$ has an $R$-successor $t \in S$, such that $\Delta_S \models C(t)$, then clearly $\Delta_S \models (\exists R.C)(s)$, because $R(s,t) \in \Delta_S$. As before, if there is no such $t$, then $M_S$ is a counterexample model to the entailment.
- For an atom of the form $(C_1 \sqcap C_2)(s)$, it follows trivially that it is entailed by $\Delta_S$ if and only if $\Delta_S \models C_1(s)$ and $\Delta_S \models C_2(s)$.
- For an atom of the form $(C_1 \sqcup C_2)(s)$, it follows trivially that it is entailed by $\Delta_S$ if $\Delta_S \models C_1(s)$ or $\Delta_S \models C_2(s)$. If neither entailments hold, then $M_S$ is a counter example to both of them, and therefore, $M_S \not\models (C_1 \sqcup C_2)(s)$.

For role atoms, if $R(s,t) \in \Delta_S$, then $\Delta_S \models R(s,t)$. If not, then $M_S$ is a counter example model.

For atoms of ordinary predicates, the atoms that can be entailed by the condition in the theorem are exactly those that would be computed in the least fixed point model and therefore satisfied in $M_S$. Every model of $\Delta_S \cup \Delta_r$ that agrees with $M_S$ on the extensions of concept and role predicates *must* satisfy at least ground atoms in the minimal fixed-point model. Hence, if a ground atom is not part of the least fixed pointed model, then it is not entailed by $\Delta_S$. $\Box$

**Proof of Lemma 6.3:** In the proof we will consider the relationship between models of $S$ and those of $S_1, \ldots, S_l$. Note that given a model $M$ of $S$, it is always possible to extend $M$ to a model $M'$ of $S \cup \Delta_r \cup \Delta_g$. That is, $M$ and $M'$ are identical on the extensions of concepts and roles, and $\alpha^{M'}$ is an extension of $\alpha^M$. Furthermore, we can consider the *unique* least fixpoint model of $S \cup \Delta_r \cup \Delta_g$. We denote the set of models of $S$ by $M(S)$. In the proof we consider one case for each propagation rule.

**Rule 1:** In this case, $l = 1$ and $S_1 = S \cup \{s : C_1, s : C_2\}$, where $s$ is the constant on which the rule was applied. It suffices to show that $M(S) = M(S_1)$. Clearly, since $S \subset S_1$, $M(S) \supseteq M(S_1)$. Let $M$ be a model of $S$. Since $M \models s : C_1 \sqcap C_2$, then $M \models s : C_1$ and $M \models s : C_2$. Therefore $M \in M(S_1)$, and hence $M(S) = M(S_1)$.

**Rules 2 and 5:** In these cases, $l = 2$, and the claim follows from the observation that $M(S) = M(S_1) \cup M(S_2)$.

**Rule 3:** Suppose $s$ has $m$ $R$-successors in $S$, $v_1, \ldots, v_m$. In this case, $l$ is $m+1$. For $i$, $1 \leq i \leq m$, $S_i = S \cup \{v_i : C\}$, and $S_{m+1} = S \cup \{sRv, v : C\}$, where $v$ is a new variable.

Assume $S \cup \Delta_r \cup \Delta_g \models q(\bar{a})$, and let $M$ be a model of $S_i \cup \Delta_r \cup \Delta_g$ for some $i$, $1 \leq i \leq m+1$. Since $S_i \supseteq S$, it follows that $M$ is a model of $S \cup \Delta_r \cup \Delta_g$, and therefore $M \models q(\bar{a})$.

Assume $S_i \cup \Delta_r \cup \Delta_g \models q(\bar{a})$ for all $i$, $1 \leq i \leq m+1$, and let $M$ be a model of $S \cup \Delta_r \cup \Delta_g$. We need to show that $M \models q(\bar{a})$. Since $(\exists R.C)(s) \in S$ there exists some $o \in \mathcal{O}^M$ such that $(s^M, o) \in R^M$, and $o \in C^M$. There are two possible cases. In the first case, there exists an $i$, $1 \leq i \leq m$, such that $v_i^M \in C^M$. In that case, $M$ is a model of $S_i \cup \Delta_r$, and therefore a model of $q(\bar{a})$. In the second case, $v_i^M \notin C^M$ for all $i$, $1 \leq i \leq m$. In this case, we can define a model $M'$ for $S_{m+1} \cup \Delta_r \cup \Delta_g$ as follows. The models $M$ and $M'$ differ only by extending $\alpha^M$ to $v$ by setting $\alpha^{M'}(v) = o$. Since the only siblings of $v$ are $v_1, \ldots, v_m$, all the inequalities involving $v$ are satisfied, and therefore $M'$ is a model of $S_{m+1} \cup \Delta_r \cup \Delta_g$. Furthermore, since $M' \models q(\bar{a})$ then $M \models q(\bar{a})$ (because we have not changed the extensions of the relations).

**Rule 4:** In this case, $l = 1$, and $S_1 = S \cup \{sRy_i \mid i \in 1..n_0\} \cup \{y_i \neq y_j \mid 1 \leq i, j, \leq n_0, i \neq j\} \cup \{y_i \neq x \mid x \in s_R, 1 \leq i \leq n_0\}$, where $y_1, \ldots, y_{n_o}$ are new variables, and $n = n_0 + m$. Clearly, since $S \subseteq S_1$, then $M(S) \supseteq M(S_1)$, and therefore, if $S \cup \Delta_r \cup \Delta_g \models q(\bar{a})$ then $S_1 \cup \Delta_r \cup \Delta_g \models q(\bar{a})$. For the other direction, assume $S_1 \cup \Delta_r \cup \Delta_g \models q(\bar{a})$, and let $M$ be a model of $S$. We need to show that $M \models q(\bar{a})$. As before, we show that we can find a model $M'$ of $S_1$ that differs from $M$ only by extending $\alpha^M$ to $y_1, \ldots, y_{n_0}$ (but the extensions of the relations do not change). Since $M \models (\geq n\, R)(s)$, there are $n$ objects, $o_1, \ldots, o_n \in \mathcal{O}^M$ such that $(s^M, o_i) \in R^M$. However, since $s$ has exactly $m$ $R$-successors in $S$, we can find a 1-1 mapping $\psi$ from $y_1, \ldots, y_{n_0}$ to $o_1, \ldots, o_n$, such that if $t$ is an $R$-successor of $s$ in $S$, then $\alpha^M(t) \neq \psi(y_i)$ for $i$, $1 \leq i \leq n_0$. Therefore, all the disequalities that are added to $S$ in $S_1$ are satisfied if we take $\psi$ to be the extension of $\alpha^M$. $\Box$

# References


[1] Andreas Abecker and Holger Wache. A layer architecture for the integration of rules, inheritance, and constraints. In *Proceedings of ICLP 94 post conference workshop on the Integration of Declarative Paradigms*, 1994.

[2] Hasan Aït-Kaci and Andreas Podelski. Towards the meaning of LIFE. In J. Maluszyński and M. Wirsing, editors, *Proceedings of the 3rd International Symposium on Programming Language Implementation and Logic Programming (Passau, Germany)*, pages 255–274. Springer Verlag, 1991.

[3] Yigal Arens, Craig A. Knoblock, and Wei-Min Shen. Query reformulation for dynamic information integration. *International Journal on Intelligent and Cooperative Information Systems*, (6) 2/3:99–130, June 1996.

[4] F. Baader, M. Buchheit, and B. Hollunder. Cardinality restrictions on concepts. *Artificial Intelligence*, 88(1-2):195–213, 1996.

[5] 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.

[6] Catriel Beeri, Alon Y. Levy, and Marie-Christine Rousset. Rewriting queries using views in description logics. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Tucson, Arizona.*, 1997.

[7] Alexander Borgida. On the relationship between description logic and predicate logic. In *The Proceedings of the third International Conference on Information and Knowledge Management (CIKM-94)*, 1994.

[8] R. J. Brachman, A. Borgida, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Resnick. Living with CLASSIC: When and how to use a KL-ONE-like language. In John Sowa, editor, *Principles of Semantic Networks*, pages 401–456. Morgan Kaufmann, San Mateo, CA, 1991.

[9] Ronald J. Brachman, Victoria P. Gilbert, and Hector J. Levesque. An essential hybrid reasoning system: Knowledge and symbol level accounts of krypton. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1985.

[10] M. Buchheit, M. Jeusfeld, W. Nutt, and M. Staudt. Subsumption between queries to object-oriented databases. *Information Systems*, 19(1):33–54, 1994.

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

[12] Hans-Jurgen Burckert. A resolution principle for constrained logics. *Artificial Intelligence*, 66:235–271, 1994.

[13] Marco Cadoli, Luigi Palopoli, and Maurizio Lenzerini. Datalog and description logics: Expressive power. In *Proceedings of the International Workshop on Database Programming Languages*, 1997.

[14] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. A hybrid system with datalog and concept languages. In E. Ardizzone, S. Gaglio, and F. Sorbello, editors, *Trends in Artificial Intelligence*, volume LNAI 549, pages 88–97. Springer Verlag, 1991.

[15] Francesco. M. Donini, Maurizion Lenzerini, Daniele Nardi, and Werner Nutt. The complexity of concept languages. In *Proceedings of KR-91*, 1991.

[16] M. Friedman and D. Weld. Efficient execution of information gathering plans. In *Proceedings of the International Joint Conference on Artificial Intelligence, Nagoya, Japan*, 1997.

[17] Alan Frisch, editor. *Working Notes of the AAAI Fall Symposium on Principles of Hybrid Reasoning*. American Association for Artificial Intelligence., 1991.

[18] 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.

[19] Philipp Hanschke and Knut Hinkelman. Combining terminological and rule-based reasoning for abstraction processes. DFKI Research Report, 1992.

[20] B. Hollunder and F. Baader. Qualifyng number restrictions in concept languages. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, 1991.

[21] 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.

[22] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query answering algorithms for information agents. In *In Proceedings of AAAI*, 1996.

[23] 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.

[24] 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.

[25] Alon Y. Levy and Marie-Christine Rousset. Verification of knowledge bases using containment checking. In *In Proceedings of AAAI*, 1996.

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

[27] Alon Y. Levy and Yehoshua Sagiv. Semantic query optimization in datalog programs. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Jose, CA*, pages 163–173, 1995.

[28] R. M. MacGregor. A deductive pattern matcher. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 403–408, 1988.

[29] Robert M. MacGregor. A description classifier for the predicate calculus. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 213–220, 1994.

[30] Margarida Mamede and Luis Monteiro. A constraint logic programming scheme for taxonomic reasoning. In *Proceedings of the International Conference on Logic Programming*, 1992.

[31] C. Petalson. The BACK system : an overview. In *Proceedings of the SIGART bulletin*, volume 2(3), pages 114–119, 1991.

[32] 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.

[33] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[34] Gert Smolka. Feature constraint logics for unification grammars. *Journal of Logic Programming*, 12:51–87, 1992.

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

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

[37] W. Wahlster, E. Andre, W. Finkler, H.J. Profitlich, and T. Rist. Plan-based integration of natural language and graphics generation. *Artificial Intelligence*, 63(1-2):387–428, 1993.

[38] J.R Wright, E.S. Weixelbaum, K. Brown, G.T Vesonder, S.R Palmer, J.I Berman, and H.H Moore. A knowledge-based configurator that supports sales, engineering and manufacturing at AT&T network systems. In *Proceedings of the Conference on Innnovative Applications of Artificial Intelligence Conference*, pages 183–193, 1993.

[39] John Yen. A principled approach to reasoning about the specificity of rules. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 701–707, 1990.