Theoretical Computer Science 52 (1987) 281-306
North-Holland

DOMAIN-INDEPENDENT FORMULAS AND DATABASES

Rodney W. TOPOR
Department of Computer Science, University of Melbourne, Parkville, Vic. 3052, Australia

Communicated by J. Ullman
Received January 1986
Revised June 1987

Abstract. This paper studies the class of deductive databases for which the result of query evaluation is independent of the domains of variables in database clauses. We first describe the class of domain-independent formulas in proof-theoretic terms. A new recursive subclass of 'allowed' formulas is defined and its properties studied. It is shown that every allowed formula is domain-independent and that every domain-independent formula has an (almost) equivalent allowed formula.

We then motivate and define the new concept of a domain-independent (deductive) database. We define a recursive class of 'allowed' databases, prove that two important subclasses of the allowed databases are domain-independent and that every domain-independent database has an equivalent allowed database (in a certain sense), describe when domain-closure axioms are unnecessary, and show that a natural query evaluation process for allowed formulas and databases has desirable properties.

Key words. Database management, deductive database, domain-independent formula, domain-independent database, allowed database, logic programming.

1. Introduction

The standard logical view of a relational database is that of a particular *interpretation* for a first-order language. In this *model-theoretic* view, query evaluation is the process of finding the tuples for which a query (a first-order formula) is true with respect to given interpretation.

It has long been recognized, however, that only certain formulas make 'reasonable' queries in this setting [8, 10]. Informally, a query is only regarded as 'reasonable' if it yields the same answer whatever the domain of the interpretation. Such queries have thus been called 'domain-independent', and have been extensively studied in, for example, [18]. A simple example of an 'unreasonable' query is the formula $\forall x\, p(x)$, whose truth depends on the domain of the variable x.

Recently, however, Reiter [19] and others have argued persuasively that it is more fruitful to view a database in *proof-theoretic* terms. In this view, a database is regarded as a particular first-order *theory*, and query evaluation is the process of

finding the instances of a query that can be proved from the theory. This approach offers many advantages for the representation of incomplete information, null values, and other real-world semantics.

Our first aim in this paper is to present a proof-theoretic characterization of the domain-independent formulas. We also introduce a new recursive subclass of the domain-independent formulas, the 'allowed' formulas, prove that every domain-independent formulas has an (almost) equivalent allowed formula, and compare the allowed formulas with other subclasses of the domain-independent formulas.

There has also been considerable interest recently in databases that contain general rules for deriving implicit information in addition to explicit facts [9, 11]. In such *deductive* databases, query evaluation is the process of proving theorems from the general rules and explicit facts. If a database contains general rules, the relevant first-order theory can have several models, so only the proof-theoretic view of a database is applicable. Introductions to the theory of deductive databases are given in [9, 14, 15].

An equivalent approach that restricts attention to relational databases but allows queries containing virtual relations and the possibly recursive general rules that define them has also been advocated recently [2, 24].

Our second aim in this paper is to observe that only certain deductive databases may be regarded as 'reasonable', and to extend the concept of domain-independent formulas to domain-independent databases. We start by defining the class of domain-independent databases (in proof-theoretic terms) and investigating their properties. We then introduce a new class of databases, the 'allowed' databases, prove that two important subclasses of the allowed databases are domain-independent and that every domain-independent database has an equivalent allowed database (in a certain sense), and investigate their other properties. We briefly discuss the role of domain-closure axioms in database theory and state conditions under which they are unnecessary. Finally, we show that allowed databases have desirable operational properties.

The structure of this paper is as follows. Section 2 contains definitions of the basic concepts of (deductive) databases. Section 3 describes domain-independent and allowed formulas and their properties. Section 4 defines domain-independent and allowed databases and studies their properties. Section 5 summarizes our results and presents our suggestions for future research.

We assume familiarity with the basic theory of logic programming, which can be found in [12]. The notation and terminology of this paper is consistent with [12, 14, 15].

2. Basic concepts

In this section, we introduce the concepts of a deductive database and query. We give the definition of the completion of a database and a correct answer. We also

define several important classes of databases and briefly describe a natural query evaluation process.

Each database and query is assumed to be expressed in some first-order function-free language with equality. In practice, typed languages should be used, but, for simplicity, we also restrict our attention to type-free languages throughout. We assume that each such language contains only finitely many constants and predicates, and at least one constant. We say that a language L_1 is *less than* another language L_2 ($L_1 \leq L_2$) if L_2 extends L_1 [21], that is, if L_2 contains at least all the constants and predicates in L_1. $\forall(W)$ denotes $\forall x_1 \ldots \forall x_n\, W$, where $x_1, \ldots, x_n$ are the free variables in the formula W.

The concepts of interpretation, model, Herbrand model, logical consequence, and so on, are defined in the usual way for first-order theories. In the presence of the equality axioms given below, by [16, p. 83], we can assume throughout that every interpretation considered is *normal*, that is, equality is always assigned the identity relation.

One of our main aims is to study the way in which the set of correct answers to a query depends on the particular language used.

Definition. A *database clause* is a first-order formula of the form $A \leftarrow W$, where A is an atom $p(t_1, \ldots, t_n)$, p is not $=$, and W is a first-order formula. A is called the *head* and W the *body* of the clause. The formula W may be absent. Any variables in A and any free variables in W are assumed to be universally quantified at the front of the clause.

Definition. A *database* is a finite set of database clauses. A database D is in *general form* if every database clause has the form $A \leftarrow L_1 \wedge \cdots \wedge L_n$, where $L_1, \ldots, L_n$ are literals.

A (nondeterministic) algorithm for transforming a given database D into a corresponding database D' in general form, called a *general form of D*, is given in [13]. Each such D' is equivalent to D in the sense given by Lemma 2.1 below. General forms of a database are important both for query evaluation and for proving properties of the database.

Definition. A *query* is a first-order formula of the form $\leftarrow W$, where W is a first-order formula, and any free variables in W are assumed to be universally quantified at the front of the query.

Definition. Let D be a database, Q a query $\leftarrow W$, and $x_1, \ldots, x_n$ the free variables in W. An *answer* for $D \cup \{Q\}$ is a substitution for some or all of the variables $x_1, \ldots, x_n$.

To define a correct answer, we first introduce the completion of a database with respect to a language.

Definition. Let D be a database in language L and p a predicate occurring in L. Suppose predicate p has definition

$$A_1 \leftarrow W_1$$
$$\vdots$$
$$A_k \leftarrow W_k$$

in D, where each A_i has the form $p(t_1, \ldots, t_n)$ for some terms $t_1, \ldots, t_n$. Then the *completed definition* of p is the formula

$$\forall x_1 \ldots \forall x_n (p(x_1, \ldots, x_n) \leftrightarrow E_1 \vee \cdots \vee E_k),$$

where $x_1, \ldots, x_n$ are variables not appearing in any $A_i \leftarrow W_i$, each E_i has the form

$$\exists y_1 \ldots \exists y_d (x_1 = t_1 \wedge \cdots \wedge x_n = t_n \wedge W_i),$$

and $y_1, \ldots, y_d$ are the variables in A_i and the free variables in W_i. If there is no clause in D with predicate p in its head, then the *completed definition* of p is the formula

$$\forall x_1 \ldots \forall x_n \sim p(x_1, \ldots, x_n).$$

Definition. The *equality theory* for a language L consists of all axioms of the following form:

(1) $c \neq d$, where c and d are distinct constants in L;
(2) $\forall x\ (x = x)$;
(3) $\forall((x_1 = y_1 \wedge \cdots \wedge x_n = y_n) \rightarrow (p(x_1, \ldots, x_n) \rightarrow p(y_1, \ldots, y_n)))$, where p (including $=$) is a predicate in L;
(4) $\forall x\ (x = c_1 \vee \cdots \vee x = c_n)$, where $c_1, \ldots, c_n$ are all the constants in L.

Axioms (1) to (3) are the usual equality axioms for a function-free program [12]. Axiom (4) is the *domain-closure axiom, DCA* [19].

Definition. Let D be a database and L a language extending that of D. The *completion* of D w.r.t. L, denoted $\text{comp}_L(D)$, is the collection of completed definitions for each predicate in L together with the above equality theory for L. We write $\text{comp}(D)$ for the completion of D when we are not concerned with the particular language involved.

The equality theory for L constrains each model for $\text{comp}_L(D)$ to be a Herbrand interpretation for L. Each model for $\text{comp}_L(D)$ can thus be identified with a subset of the Herbrand base B_L of the ground atoms in L.

We can now define a correct answer.

Definition. Let D be a database, Q a query $\leftarrow W$, and L a language extending that of D. A *correct answer* for $\text{comp}_L(D) \cup \{Q\}$ is an answer θ such that $\forall(W\theta)$ is a logical consequence of $\text{comp}_L(D)$.

By the completeness of first-order logic, θ is a correct answer for $\text{comp}_L(D) \cup \{Q\}$ if and only if $\forall(W\theta)$ can be proved from $\text{comp}_L(D)$. The concept of a correct answer gives a declarative description of the desired output from a query to a database.

It is convenient to introduce the following definition at this point.

Definition. Let D be a database, W a formula, and L a language extending that of D and W. Then $\text{ans}(D, W, L)$ is the set of all correct answers for $\text{comp}_L(D) \cup \{\leftarrow W\}$ that are ground substitutions for all free variables in W.

We can now also describe the sense in which a general form of a database is equivalent to the database.

Lemma 2.1 (Lloyd and Topor [13, 15]). *Let D be a database, D' a general form of D, and W a closed formula that only contains predicates in D. Then W is a logical consequence of* $\text{comp}(D)$ *if and only if W is a logical consequence of* $\text{comp}(D')$.

Lemma 2.1 states that $\text{comp}(D')$ is a conservative extension of $\text{comp}(D)$ [21].

We now define several important classes of databases.

Definition. A database is *relational* if every database clause has the form $p(c_1, \ldots, c_n) \leftarrow$, where $c_1, \ldots, c_n$ are constants.

Definition. A database is *definite* [14] if every database clause has the form $A \leftarrow A_1 \wedge \cdots \wedge A_m$, where $A_1, \ldots, A_m$ are atoms. A query is *definite* if it has the form $\leftarrow A_1 \wedge \cdots \wedge A_m$, where $A_1, \ldots, A_m$ are atoms.

Clearly, every relational database is definite.

Definition. A *level mapping* of a database is a mapping from its set of predicates to the natural numbers. We refer to the level of a predicate under the mapping as the *level* of the predicate.

Definition. A database is *hierarchical* [3, 14] if it has a level mapping such that, in every database clause $p(t_1, \ldots, t_n) \leftarrow W$, the level of every predicate in W is less than the level of p.

Without loss of generality, we will assume that the predicate levels in a hierarchical database are $0, 1, \ldots, k$, for some k.

If D is a definite or hierarchical database, then it is straightforward to prove that, for any language L extending that of D, $\text{comp}_L(D)$ has a (Herbrand) model. For other databases, the existence of a model for $\text{comp}_L(D)$ can depend on L.

Example. Let D be the database

$$p(a) \leftarrow \exists x \sim p(x)$$

in language L. The completed definition of p is

$$p(a) \leftrightarrow \exists x \sim p(x).$$

If the only constant in L is a, then there is no model for $\text{comp}_L(D)$, but if L also contains another constant, b say, then $\{p(a)\}$ is a model for $\text{comp}_L(D)$.

We conclude this section by summarizing a natural query evaluation process for deductive databases described in [15]. Let D be a database and Q a query $\leftarrow W$. First, replace D by $D' = D \cup \{\text{answer}(x_1, \ldots, x_n) \leftarrow W\}$ and Q by $\leftarrow \text{answer}(x_1, \ldots, x_n)$, where $x_1, \ldots, x_n$ are the free variables in W and 'answer' is a new predicate. Second, replace D' by a general form D'' of D' using the algorithm given in [13]. Third, call each computed answer for $D'' \cup \{\leftarrow \text{answer}(x_1, \ldots, x_n)\}$ found by SLDNF-resolution [12], i.e., by a Prolog system which only selects a negative literal if it is ground, a normal computed answer for $D \cup \{Q\}$. A selected literal $s = t$ succeeds if s and t are unifiable. This process is called *normal query evaluation* in [15], where its soundness and completeness properties are presented.

3. Domain-independent formulas

We begin this section with the standard definition of a domain-independent formula.

Definition. Let W be a formula with k free variables and L a language extending that of W. Let I be an interpretation for L with domain U. If $k > 0$, $\text{val}(W, I)$ is the set of elements of U^k for which W is true w.r.t. I. If $k = 0$, $\text{val}(W, I)$ is *true* or *false* depending on whether or not W is true w.r.t. I.

Definition. A formula W in language L is (model-theoretically) *domain-independent* if, for all finite interpretations I_1 and I_2 for L that assign the same domain elements to the constants in W and the same relations to the predicates in W, $\text{val}(W, I_1) = \text{val}(W, I_2)$.

That is, W is (model-theoretically) domain-independent if the set of tuples for which W is true w.r.t. I depends only on the domain elements assigned by I to the constants in W and on the relations assigned by I to the predicates in W, but not on the domains of the variables in W. The value of a domain-independent formula is thus invariant (or 'stable' [18]) with respect to database (or interpretation) updates that do not affect relations in the formula.

Note that domain-independence of a formula is a model-theoretic property and the domain-independence of a closed formula is thus preserved under transformations that preserve logical equivalence.

Example. Clearly, every valid closed formula is domain-independent. The following formulas are also domain-independent:

$$p(x), \qquad \exists x \exists y (p(x) \vee q(y)),$$

$$\exists x \forall y (p(y) \rightarrow q(x, y)).$$

The following formulas are not domain-independent:

$$\sim p(x), \qquad \exists y (p(x) \vee q(y)),$$

$$\forall y (p(y) \rightarrow q(x, y)), \qquad \exists x p(x) \wedge \exists x \sim p(x).$$

For each formula W that is not domain-independent, val(W, I) depends on the domain of I, and W thus cannot be considered a 'reasonable' query.

Next we give our proof-theoretic definition of a domain-independent formula. It is given here as an example of how standard concepts can be described in these terms and as an introduction to the proof-theoretic definition of domain-independent databases given in the next section.

Definition. A formula W is (proof-theoretically) *domain-independent* if, for all languages L_1 and L_2 extending that of W and for all relational databases D in L_1 and L_2, $\text{ans}(D, W, L_1) = \text{ans}(D, W, L_2)$.

Informally, W is (proof-theoretically) domain-independent if, for every relational database D in a language L_D extending that of W and for every language L extending L_D, the set of correct answers for $\text{comp}_L(D) \cup \{\leftarrow W\}$ is independent of L.

The following theorem states that the above two definitions of domain-independence describe the same class of formulas.

Theorem 3.1. *Let W be a formula in language L. Then W is (model-theoretically) domain-independent if and only if W is (proof-theoretically) domain-independent.*

Proof. (*Only if*): Suppose that W is (model-theoretically) domain-independent. Let L_1 and L_2 be languages extending that of W, D a relational database in L_1 and L_2, and θ an answer in $\text{ans}(D, W, L_1)$. By [19, Theorem 3.1(1)], $\text{comp}_{L_1}(D)$ has a unique normal (Herbrand) model I_1, and $W\theta$ is true w.r.t. I_1. Let I_2 be a normal Herbrand interpretation for L_2 that assigns the same relations to predicates of D as I_1. Then, by assumption, $W\theta$ is true w.r.t. I_2. But I_2 is the unique normal model for $\text{comp}_{L_2}(D)$, so θ is in $\text{ans}(D, W, L_2)$.

(*If*): Suppose that W is (proof-theoretically) domain-independent. Let I_1 and I_2 be finite normal interpretations for L that assign the same domain elements to the constants in W and the same relations to the predicates in W. Let $x_1, \ldots, x_m$ be the free variables in W and suppose that $\langle d_1, \ldots, d_m \rangle$ belongs to val(W, I_1). Let L_i be a language with constants corresponding to the domain elements of I_i for $i = 1, 2$. By [19, Theorem 3.1(2)], there exists a single relational database D in L_i such that I_i is the unique normal model for $\text{comp}_{L_i}(D)$ for $i = 1, 2$. Let θ be the substitution $\{x_1/d_1, \ldots, x_m/d_m\}$. As I_1 is the unique normal model for $\text{comp}_{L_1}(D)$, θ is in ans(D, W, L_1). Thus, by assumption, θ is in ans(D, W, L_2). It follows that $\langle d_1, \ldots, d_m \rangle$ belongs to val(W, I_2). □

Unfortunately, the class of domain-independent formulas has the following property.

Theorem 3.2 (Di Paola [8], Vardi [26]). *The decision problem for the class of domain-independent formulas is recursively unsolvable.*

As a result of Theorem 3.2, various recursive subclasses of the domain-independent formulas have been proposed. These include the *range-separable* formulas [4], the *range-restricted* formulas [17], and the *evaluable* formulas [6, 7]. Properties of these classes of formulas are discussed below.

Another important subclass of the domain-independent formulas is the class of 'safe' formulas, proposed by Ullman in [23] to ensure that val(W, I) is finite even for those interpretations I with an infinite domain.

Definition. Let W be a formula in language L and I an interpretation for L. Then dom(W, I) is the set of domain elements that are assigned by I to constants in W or are components of tuples in relations that are assigned by I to predicates in W.

Definition. A formula $W(x_1, \ldots, x_n)$ in language L is *safe* if, for all interpretations I for L, the following conditions hold:

(1) if $W(d_1, \ldots, d_n)$ is true w.r.t. I, then each d_i is in dom(W, I);
(2) if $\exists u\, F(u)$ is a subformula of W, then $F(d)$ true w.r.t. I for some values of the free variables in F (besides u) implies that d is in dom(W, I);
(3) if $\forall u\, F(u)$ is a subformula of W, then $F(d)$ false w.r.t. I for some values of the free variables in F (besides u) implies that d is in dom(W, I).

Some examples of safe formulas will be given later. First we note the following result.

Lemma 3.3. *Every safe formula is domain-independent.*

Proof. The following proof due to Nicolas and Demolombe [18] is included for completeness. Let W be a safe formula in language L, and I_1 and I_2 finite interpretations for L that assign the same domain elements to the constants in W and the

same relations to the predicates in W. By [23, Theorem 5.3], W has an equivalent relational algebra expression A containing no new constants or predicates; that is, $\text{val}(W, I_i) = \text{aval}(A, I_i)$ for $i = 1, 2$, where $\text{aval}(A, I)$ is the value of the relational algebra expression A in interpretation I. But $\text{aval}(A, I_1) = \text{aval}(A, I_2)$ as $\text{aval}(A, I)$ depends only on the domain elements assigned by I to the constants in A and the relations assigned by I to the predicates in A. Thus, $\text{val}(W, I_1) = \text{val}(W, I_2)$. □

The converse of Lemma 3.3 is false.

Example. The following formulas are domain independent but not safe:

$$\exists x(p(x) \vee q(a)),$$

$$\exists x \sim p(x) \vee \forall y\, p(y).$$

We now define another recursive subclass of the domain-independent formulas: the 'allowed' formulas. We will see that this class has a combination of desirable properties not possessed by any of the above classes of formulas.

Definition. A variable x is *pos* (positive) in a formula W if one of the following cases holds.

- x is pos in $p(t_1, \ldots, t_n)$ if x occurs in $p(t_1, \ldots, t_n)$ and p is not $=$;
- x is pos in $x = c$ or $c = x$ if c is a constant;
- x is pos in $\sim F$ if x is neg in F;
- x is pos in $F \wedge G$ if x is pos in F or x is pos in G;
- x is pos in $F \vee G$ if x is pos in F and x is pos in G;
- x is pos in $F \to G$ if x is neg in F and x is pos in G;
- x is pos in $\exists y\, F$ if x is pos in F.

Similarly, x is *neg* (negative) in W if one of the following cases holds.

- x is neg in $\sim F$ if x is pos in F;
- x is neg in $F \wedge G$ if x is neg in F and x is neg in G;
- x is neg in $F \vee G$ if x is neg in F or x is neg in G;
- x is neg in $F \to G$ if x is pos in F or x is neg in G;
- x is neg in $\forall y\, F$ if x is neg in F.

Definition. A formula W is *allowed* if the following conditions hold:

(1) every free variable in W is pos in W;
(2) for every subformula $\exists x\, F$ of W, x is pos in F;
(3) for every subformula $\forall x\, F$ of W, x is neg in F.

In allowed formulas, atoms containing the positive occurrence of x in $\exists x\, F$ or the negative occurrence of x in $\forall x\, F$ act as generators of instances of x during normal query evaluation.

Example. The following formulas are allowed:

$$p(x),$$

$$p(x, y) \wedge \sim q(x),$$

$$p(x) \wedge \forall y(q(y) \rightarrow r(x, y)),$$

$$\exists x \exists y(p(x) \wedge \sim(q(y) \rightarrow \forall z \sim r(x, y, z))).$$

Examples of some formulas that are not allowed will be given later.

Lemma 3.4. *Every allowed formula is safe.*

Proof. Let W be an allowed formula in language L, I a finite normal interpretation for L, and V a variable assignment that maps variable x to domain element d. It is straightforward to prove by induction on W that x pos in W and W true w.r.t. I and V implies that d belongs to dom(W, I) and x neg in W and W false w.r.t. I and V implies that d belongs to dom(W, I). That W is safe then follows. □

The converse of Lemma 3.4 is false.

Example. The following formula is safe but not allowed:

$$\exists x \exists y(p(x) \wedge x = y \wedge \sim q(y)).$$

An important property of allowed formulas is expressed in the following theorem.

Theorem 3.5. *Every allowed formula is domain-independent.*

Proof. This follows directly from Lemmas 3.3 and 3.4. □

Corollary 3.6. *Every formula of the form $A_1 \wedge \cdots \wedge A_m$, where $A_1, \ldots, A_m$ are atoms, is domain-independent.*

Proof. Immediate. □

The converse of Theorem 3.5 is false. Clearly, as the class of allowed formulas is recursive and the class of domain-independent formulas is recursively unsolvable, there are domain-independent formulas that are not allowed. For example, the tautology given earlier, $\exists x \sim p(x) \vee \forall y\, p(y)$, is domain-independent but not allowed. Other examples are given later.

We note the following relationships between the allowed formulas and the other classes of domain-independent formulas. The range-separable, range-restricted, and evaluable formulas cannot contain the equality predicate, and range-restricted formulas are in prenex conjunctive normal form. In [6, 7], it is proved that every range-separable or range-restricted formula is evaluable and that every evaluable formula is domain-independent.

The definition of allowed formulas is similar to that of the evaluable formulas, but is much simpler. It is straightforward to prove by induction that every allowed formula (without equality) is evaluable. Not every evaluable formula, however, is allowed.

Example. The following formulas are evaluable but not allowed:

$$\exists x \exists y(p(x) \vee q(y)), \qquad \exists x \forall y(p(y) \rightarrow q(x, y)).$$

Similarly, it is immediate that every allowed formula in prenex conjunctive normal form is range-restricted. Again, however, the converse is false.

Example. The following formula in prenex conjunctive normal form is range restricted but not allowed:

$$\exists x \forall y(\sim p(y) \vee q(x, y)).$$

The safe formulas, as defined in [23], may contain arithmetic comparison operators $<$, $\leqslant$, etc. We have omitted such operators from our treatment for simplicity. The safe formulas are very similar to the allowed formulas, however. It may be possible to modify the definition of the class of allowed formulas so that, in the absence of arithmetic comparison operators other than equality, every safe formula is allowed, but this would destroy the simple compositional nature of the current definition.

Other relationships between these classes of formulas are investigated in [6, 7, 18, 25]. If all the definitions are modified (if necessary) so that all formulas can contain equality but no other arithmetic comparison operators, then the relationships we have discussed may be summarized as follows:

allowed $\rightarrow$ safe $\rightarrow$ domain-independent,

allowed $\rightarrow$ evaluable $\rightarrow$ domain-independent,

range-separable $\rightarrow$ evaluable.

Each of these implications is strict. For formulas in prenex conjunctive or disjunctive normal form, the following relationship holds.

evaluable $\leftrightarrow$ range-restricted.

Despite the apparent lack of expressive power of allowed formulas in comparison with evaluable, safe, and domain-independent formulas, we can prove that every domain-independent formula is equivalent to an allowed formula in the following sense.

Theorem 3.7. *For every domain-independent formula W in language L, there exists an allowed formula W' in L such that, for every finite interpretation I for L with* $\mathrm{dom}(W, I) \neq \emptyset$, $\mathrm{val}(W, I) = \mathrm{val}(W', I)$.

Proof. The following proof is based on that of [18, Theorem 3]. Let $c_1, \ldots, c_m$ be the constants in W and $p_1, \ldots, p_r$ the predicates in W. Let $\mathrm{DOM}_W(x)$ denote the formula

$$x = c_1 \vee \cdots \vee x = c_m \vee D_1(x) \vee \cdots \vee D_r(x),$$

where each $D_j(x)$ denotes the formula

$$\exists u_1 \ldots \exists u_{k-1} p_j(x, u_1, \ldots, u_{k-1}) \vee \cdots \vee \exists u_1 \ldots \exists u_{k-1} p_j(u_1, \ldots, u_{k-1}, x).$$

For every finite interpretation I for L, $\mathrm{DOM}_W(d)$ is true w.r.t. I iff d is in $\mathrm{dom}(W, I)$. We now construct W' from W as follows:

(1) for each free variable x_i in W that is not pos in W, add the conjunct $\mathrm{DOM}_W(x_i)$ to W;
(2) replace each subformula $\exists u F$ of W in which u is not pos in F by $\exists u(\mathrm{DOM}_W(u) \wedge F)$;
(3) replace each subformula $\forall u F$ of W in which u is not neg in F by $\forall u(\mathrm{DOM}_W(u) \rightarrow F)$.

Clearly, the resulting formula W' is allowed. Also, it is straightforward to prove that, for every finite interpretation I whose domain is exactly $\mathrm{dom}(W, I)$, $\mathrm{val}(W, I) = \mathrm{val}(W', I)$.

Now, suppose that $\mathrm{dom}(W, I) \neq \emptyset$. Let I' be the interpretation for L obtained by restricting the domain of I to $\mathrm{dom}(W, I)$ and deleting from I all tuples with components not in $\mathrm{dom}(W, I)$. Then I' assigns the same domain elements to $c_1, \ldots, c_m$ and the same relations to $p_1, \ldots, p_r$ as I, so $\mathrm{dom}(W, I') = \mathrm{dom}(W, I)$. Since W is domain-independent, $\mathrm{val}(W, I) = \mathrm{val}(W, I')$. Since W' is allowed and hence domain-independent, $\mathrm{val}(W', I) = \mathrm{val}(W', I')$. Since the domain of I' is $\mathrm{dom}(W, I')$, $\mathrm{val}(W, I') = \mathrm{val}(W', I')$. Thus, finally, $\mathrm{val}(W, I) = \mathrm{val}(W', I)$. □

The condition that $\mathrm{dom}(W, I) \neq \emptyset$ in Theorem 3.7 is necessary.

Example (Nicolas and Demolombe [18]) Let W be the domain-independent formula $\forall x(p(x) \wedge \sim p(x))$, which is false w.r.t. every interpretation. Then, W' is the formula $\forall x(p(x) \rightarrow p(x) \wedge \sim p(x))$. As $\mathrm{dom}(W, I) = \emptyset$ iff $\forall x \sim p(x)$ is true w.r.t. I, W' is true w.r.t. interpretations I such that $\mathrm{dom}(W, I) = \emptyset$.

Theorem 3.7 indicates that, in all practical cases, we lose no expressive power by restricting attention to allowed formulas. The given construction for W' has the property that if W is already allowed, then W' is W. In other cases, it produces impractically large formulas and it is desirable to find alternative constructions that produce simpler formulas.

The following construction, due primarily to Decker [5], provides a solution to this problem for a large class of domain-independent formulas. Another approach to the problem is presented in [25].

Definition. Let cnf(W) (respectively dnf(W)) be the prenex conjunctive (respectively disjunctive) normal form of a formula W, constructed by moving negations inwards, moving quantifiers outwards, and applying appropriate distributive laws. Then a *prenex conjunctive* (respectively *disjunctive*) *simplified normal form* csf(W) (respectively dsf(W)) of W is a formula constructed from cnf(W) (respectively dnf(W)) by first applying all possible propositional simplifications of the form

$$P \vee \text{false} \Rightarrow P, \qquad P \wedge \text{false} \Rightarrow \text{false},$$

$$P \vee \text{true} \Rightarrow \text{true}, \qquad P \wedge \text{true} \Rightarrow P,$$

$$P \vee P \Rightarrow P, \qquad P \wedge P \Rightarrow P,$$

$$P \vee (P \wedge Q) \Rightarrow P, \qquad P \wedge (P \vee Q) \Rightarrow P,$$

$$P \vee \sim P \Rightarrow \text{true}, \qquad P \wedge \sim P \Rightarrow \text{false},$$

$$P \vee (\sim P \wedge Q) \Rightarrow P \vee Q, \qquad P \wedge (\sim P \vee Q) \Rightarrow P \wedge Q,$$

$$(P \wedge Q) \vee (\sim P \wedge Q \wedge R) \Rightarrow (P \wedge Q) \wedge (Q \wedge R),$$

$$(P \vee Q) \wedge (\sim P \vee Q \vee R) \Rightarrow (P \vee Q) \wedge (Q \vee R),$$

where P and Q are (conjunctions or disjunctions of) literals, and then omitting any quantifiers whose variables no longer occur in the matrix of the formula.

Definition. Let W be a formula such that

$$\text{csf}(W) = Q_1 x_1 \ldots Q_k x_k (C_1 \wedge \cdots \wedge C_m)$$

and

$$\text{dsf}(W) = Q_1 x_1 \ldots Q_k x_k (D_1 \vee \cdots \vee D_n),$$

where each Q_i is either $\forall$ or $\exists$. Then W is *range-restricted* [5] if the following conditions hold:

(1) every free variable in W is pos in each D_i;
(2) every existentially quantified variable in dsf(W) is pos in each D_j in which it occurs;
(3) every universally quantified variable in csf(W) is neg in each C_k in which it occurs.

This definition generalizes the one in [17].

Example. The following non-allowed formulas are range-restricted:

$$W_1 = \mathrm{csf}(W_1) = \forall x(s(y) \wedge (p(x) \vee \sim q(x))),$$

$$\mathrm{dsf}(W_1) = \forall x((s(y) \wedge p(x)) \vee (s(y) \wedge \sim q(x)));$$

$$W_2 = \mathrm{csf}(W_2) = \mathrm{dsf}(W_2) = \exists x \forall y(\sim p(y) \vee q(x, y));$$

$$W_3 = \mathrm{csf}(W_3) = \exists x \forall y((\sim p(y) \vee q(x, y) \vee s(z)) \wedge (r(x, z) \vee s(z))),$$

$$\mathrm{dsf}(W_3) = \exists x \forall y((\sim p(y) \wedge r(x, z)) \vee (q(x, y) \wedge r(x, z)) \vee s(z)).$$

It can be proved that every evaluable formula (and hence every allowed formula) is range-restricted and that every range-restricted formula is domain-independent. That is, we have the following (strict) relationship:

allowed → evaluable → range-restricted → domain-independent.

Thus, the class of range-restricted formulas is the largest recursive subclass of the domain-independent formulas we have yet considered.

Definition. Let W be a formula. Then a *range form* $r(W)$ [5] of W is a formula constructed by recursively applying the following transformations to W:

(1) If W has no quantifiers, then $r(W)$ is W.

(2) If $\mathrm{dsf}(W)$ (respectively $\mathrm{csf}(W)$) contains only existential (respectively universal) quantifiers, then $r(W)$ is the result of moving each quantifier $\exists x$ (respectively $\forall x$) inwards to cover only the disjuncts (respectively conjuncts) of $\mathrm{dsf}(W)$ (respectively $\mathrm{csf}(W)$) in which x occurs.

(3) Suppose $\mathrm{dsf}(W) = \exists x\, Q(D_1 \vee \cdots \vee D_n)$, where Q denotes $Q_1 x_1 \ldots \forall x_j \ldots Q_k x_k$, and x occurs in D_i for $1 \leqslant i \leqslant v \leqslant n$. For each D_i, $1 \leqslant i \leqslant v$, let A_i be a positive literal containing x in D_i, and let P_i denote $\exists y_1 \ldots \exists y_s A_i$, where $y_1, \ldots, y_s$ are the variables other than x in A_i. (If no such literal exists in D_i, let P_i be false.) Then $r(W)$ is:

$$\exists x((P_1 \vee \cdots \vee P_v) \wedge r(Q(D_1 \vee \cdots \vee D_n))) \vee r(Q(D_{v+1} \vee \cdots \vee D_n)).$$

(4) Suppose $\mathrm{csf}(W) = \forall x Q(C_1 \wedge \cdots \wedge C_m)$, where Q denotes $Q_1 x_1 \ldots \exists x_j \ldots Q_k x_k$, and x occurs in C_i for $1 \leqslant i \leqslant u \leqslant m$. For each C_i, $1 \leqslant i \leqslant u$, let $\sim A_i$ be a negative literal containing x in C_i, and let P_i denote $\exists y_1 \ldots \exists y_s A_i$, where $y_1, \ldots, y_s$ are the variables other than x in A_i. (If no such literal exists in C_i, let P_i be false.) Then $r(W)$ is

$$\forall x((P_1 \vee \cdots \vee P_u) \rightarrow r(Q(C_1 \wedge \cdots \wedge C_m))) \wedge r(Q(C_{u+1} \wedge \cdots \wedge C_m)).$$

Example. Applying this algorithm to the range-restricted formulas above, and applying some obvious simplifications, leads to the following corresponding range forms.

$$r(W_1) = s(y) \wedge \forall x(p(x) \vee \sim q(x)),$$

$$r(W_2) = \exists x(\exists y\, q(x, y) \wedge \forall y(\sim p(y) \vee q(x, y))) \vee \forall y \sim p(y),$$

$$r(W_3) = \exists x(\exists z\, r(x, z) \wedge \forall y(\sim p(y) \vee q(x, y)) \wedge (r(x, z) \vee s(z))) \vee s(z).$$

Again it can be proved that every range form $r(W)$ of a range-restricted formula W has the following important properties: $r(W)$ is an allowed formula, and $r(W)$ is logically equivalent to W; see [5, 25]. Thus, this construction provides a way of transforming any range-restricted formula into an equivalent allowed formula *without* introducing the new equalities required in the proof of Theorem 3.7.

In the next section, we will attempt to extend these various results from formulas to databases.

4. Domain-independent databases

As suggested in the introduction, if D is a deductive database, comp(D) can have *no* models or *several* models. For this reason alone, it is necessary to describe database properties, such as correct answers, in proof-theoretic terms rather than model-theoretic terms. Note, however, that despite the presence of arbitrary clauses in a database D the domain-closure axiom forces every model for comp(D) to be finite.

Just as not all queries are 'reasonable', so not all deductive databases are 'reasonable'. As the completion $\text{comp}_L(D)$ for a database D can depend on the language L, the set of correct answers for a perfectly acceptable (e.g., allowed) query can also depend on L. That is, the set of correct answers for a query can depend on the domains of variables in D. We give four examples of this phenomenon below. (Note that we are not concerned here with the difference between normalized and unnormalized relations.)

Example. Let D be the database

$$p(a) \leftarrow$$

$$q(a) \leftarrow \forall x\, p(x)$$

in language L. Then $q(a)$ is a logical consequence of $\text{comp}_L(D)$ if and only if a is the only constant in L. Thus, D cannot be considered a 'reasonable' database.

Example. Let D be the database

$$p(a) \leftarrow$$

$$q(x) \leftarrow \sim p(x)$$

in language L and Q the query $\leftarrow q(x)$. Then, if a is the only constant of L, there are no correct answers for $D \cup \{Q\}$ w.r.t. L. But, if L contains any constant $b \neq a$, then $\{x/b\}$ is a correct (ground) answer for $D \cup \{Q\}$ w.r.t. L. So, again, D is not a 'reasonable' database.

Example. Let D be the database

$$p(a) \leftarrow$$

$$q(x) \leftarrow$$

$$r(x) \leftarrow q(x) \wedge \sim p(x)$$

in language L and Q the query $\leftarrow r(x)$. Then, again, if a is the only constant in L, there are no correct answers for $D \cup \{Q\}$ w.r.t. L. But, if L contains any constant $b \neq a$, then $\{x/b\}$ is a correct (ground) answer for $D \cup \{Q\}$ w.r.t. L, and D is not a 'reasonable' database.

Example. Let D be the database

$$p(a) \leftarrow$$

$$r(x, y) \leftarrow p(x) \vee q(y)$$

in language L. In this case also, the set of (ground) correct answers for $\text{comp}_L(D) \cup \{\leftarrow r(x, y)\}$ depends on L, so D is not a 'reasonable' database.

The following definition attempts to capture the concept of a 'reasonable' database.

Definition. A database D is *domain-independent* if, for all languages L_1 and L_2 extending that of D and for all atoms A in L_1 and L_2, $\text{ans}(D, A, L_1) = \text{ans}(D, A, L_2)$.

Corollary 4.1. *Let D be a database and $L \leq L'$ languages extending that of D. Then D is domain-independent if and only if, for all atoms A in L, $\text{ans}(D, A, L) = \text{ans}(D, A, L')$.*

Proof. Immediate. □

Thus, D is domain-independent if, for every atom A in a language L_A extending that of D and for every language L extending L_A, the set of correct answers for $\text{comp}_L(D) \cup \{\leftarrow A\}$ is independent of L. Equivalently, D is domain-independent if, for every language L extending that of D, the set of ground atoms that are logical consequences of $\text{comp}_L(D)$ is independent of L. The use of ground answers in the examples and definition is based on the assumption that a typical user of a database system is only interested in ground answers to a query.

Note that domain-independence of a database D is a model-theoretic property and is thus preserved under transformations which preserve the logical equivalence of $\text{comp}(D)$.

This definition appears to be the natural generalization of the proof-theoretic definition of a domain-independent formula. As both definitions state that the answers obtained are the same whichever language is used, it may have been more logical to refer to *language*-independent formulas and databases, but we have used the standard terminology for (historical) consistency. Moreover, in the presence of the domain-closure axiom, a language L determines the domain of each interpretation for L.

For this definition to be satisfactory, however, it is desirable, even necessary, for every domain-independent database D to possess the more general property that, for every reasonable (e.g., allowed) query Q in a language L_Q extending that of D and for every language L extending that of L_Q, the set of correct answers for $\text{comp}_L(D) \cup \{Q\}$ is independent of L. We discuss the extent to which domain independent databases do possess this property in Theorem 4.11 below.

Our first result asserts that every domain-independent database has a consistent completion.

Theorem 4.2. *Let D be a domain-independent database. Then, for some language L extending that of D, $\text{comp}_L(D)$ has a model.*

Proof. Suppose that, for every language L extending that of D, there is no model for $\text{comp}_L(D)$. Let p be an n-ary predicate in D, $n \geq 1$, and let L be any language extending that of D. Then, for each tuple $c_1, \ldots, c_n$ of constants in L, $\{x_1/c_1, \ldots, x_n/c_n\}$ is in $\text{ans}(D, p(x_1, \ldots, x_n), L)$. That is, $\text{ans}(D, p(x_1, \ldots, x_n), L)$ depends on L, so D is not domain-independent, which is a contradiction. □

We conjecture that, for every domain-independent database D and for *every* language L extending that of D, $\text{comp}_L(D)$ has a model.

We now consider the decision problem for the class of domain-independent databases. First, we define a mapping T_D associated with a database D from the lattice of Herbrand interpretations for a language L into itself.

Definition. Let D be a database, L a language extending that of D, and I a Herbrand interpretation for L. Then $T_D(I) = \{A\theta : A \leftarrow W$ is a clause in D, θ is a ground substitution in L for the variables in A and the free variables in W, and $W\theta$ is true w.r.t. $I\}$.

We now give an important special case of the decision problem.

Theorem 4.3. *The decision problem for the class of domain-independent definite databases is recursively solvable.*

Proof. Let D be a definite database and L a language extending that of D. By [12, Theorem 6.6], θ is in $\text{ans}(D, A, L)$ if and only if $A\theta$ is true w.r.t. the least Herbrand

model $M(D, L)$ for D w.r.t. L. Thus, it suffices to show that there is an algorithm to decide whether, for all languages L_1 and L_2 extending that of D, $M(D, L_1) = M(D, L_2)$.

Let L be the language of D and T_D the corresponding mapping. Let $n \geq 0$ be such that $T_D \uparrow \omega = T_D \uparrow n$. Then, it is straightforward to prove that there exists a language $L' > L$ with $M(D, L') \neq M(D, L)$ if and only if the following condition holds: for some k with $0 \leq k < n$, there exists an instance $(A \leftarrow A_1 \wedge \cdots \wedge A_m)\theta$ of a clause in D such that

(a) θ is a ground substitution for the variables in $(A_1 \wedge \cdots \wedge A_m)\theta$;

(b) $(A_1 \wedge \cdots \wedge A_m)\theta$ is true in $T_D \uparrow k$; and

(c) $A\theta$ is not ground.

This condition can be effectively tested, and the result follows. □

A similar argument shows that the decision problem for the class of domain-independent hierarchical databases is also solvable. We conjecture, however, that, as for the class of domain-independent formulas, the decision problem for the class of domain-independent databases is recursively unsolvable in general.

If this conjecture is true, it is clearly necessary to search for recursive subclasses of the domain-independent databases. Even if the conjecture is false, it is desirable to search for recursive subclasses with simple decision procedures. To these ends, we introduce the class of 'allowed' databases. We prove that every allowed database that is definite or hierarchical is domain-independent and that every domain-independent database has an equivalent allowed database (in a certain sense).

Definition. A database D is *allowed* if each clause in D is an allowed formula. A query $\leftarrow W$ is *allowed* if W is an allowed formula.

Example. The following database is allowed:

$$p(a) \leftarrow$$

$$q(b, x) \leftarrow p(x) \vee q(x, c)$$

$$r(x, z) \leftarrow q(x, y) \wedge r(y, z) \wedge \sim p(x)$$

$$s(x, y) \leftarrow \exists z(p(x, y, z) \wedge \forall w(q(d, w) \rightarrow r(w, y))).$$

Similar classes of databases have been proposed by Clark [3] and Shepherdson [20] for databases in general form and by Lloyd and Topor [15]. These classes are related as follows. Every general clause $p(t_1, \ldots, t_n) \leftarrow W$ that satisfies the 'covering axiom' [20] and whose body W is an allowed formula [3] is allowed (in the current sense).

Lemma 4.4. *If D is an allowed database and Q is an allowed query, then $D \cup \{Q\}$ is allowed in the sense of* [15].

Proof. The result follows by a straightforward induction on the number of transformations performed in transforming D to general form. □

The converse of Lemma 4.4 is false.

Example. The following database is allowed in the sense of [15] because the predicate q only occurs in negative literals in clause bodies. But it is not an allowed database.

$$p(a) \leftarrow$$

$$q(x) \leftarrow$$

$$r(x) \leftarrow p(x) \wedge \sim q(x).$$

We believe that the new definition is not significantly more restrictive than the one in [15] in practice. It is interesting that, although the restriction to 'allowed' databases and to databases satisfying the 'covering axiom' were introduced to obtain completeness results for query evaluation processes, they also serve to ensure domain-independence.

We now prove that two important subclasses of the allowed databases are domain-independent. This requires the following lemmas of independent interest.

Lemma 4.5. *Let D be an allowed database. Then every completed definition in* comp(D) *is safe and hence domain-independent.*

Proof. Straightforward. □

Lemma 4.6. *Let D be an allowed database, $L \leq L'$ languages extending that of D, M a (Herbrand) model for* $\text{comp}_L(D)$, *and M' the (Herbrand) interpretation for L' containing the same set of ground atoms as M. Then M' is a model for* $\text{comp}_{L'}(D)$.

Proof. The result follows directly from Lemma 4.5 and the definition of a domain-independent formula. □

Lemma 4.7. *Let D be an allowed database, $L \leq L'$ languages extending that of D, I a Herbrand interpretation for L, and I' the Herbrand interpretation for L' containing the same set of ground atoms as I.*

(a) *Let W be an allowed formula in L with free variables $x_1, \ldots, x_m$. If $W(c_1, \ldots, c_m)$ is true w.r.t. I' for some tuple $c_1, \ldots, c_m$ of constants in L', then each c_i is in L.*

(b) *Let T_D (respectively T'_D) be the mapping associated with D from the lattice of Herbrand interpretations for L (respectively L') into itself. Then $T_D(I) = T'_D(I')$.*

Proof. (a): By Lemma 3.4, W is safe. Thus, each c_i is in $\text{dom}(W, I') = \text{dom}(W, I)$ and is hence in L.

(b): Let $A \leftarrow W$ be a clause in D and $\theta = \{x_1/c_1, \ldots, x_m/c_m\}$ a ground substitution for the variables in A and the free variables in W. Then $A\theta$ is in $T_D(I)$ (respectively $T'_D(I')$) if and only if $W\theta$ is true w.r.t. I (respectively I'). By part (a), we may assume that each c_i is in L. By Theorem 3.5, W is domain-independent, so $W\theta$ is true w.r.t. I if and only if $W\theta$ is true w.r.t. I'. The result follows immediately. □

Theorem 4.8. *Every allowed definite database is domain-independent.*

Proof. Let D be an allowed definite database, $L \leqslant L'$ languages extending that of D, and A an atom in L. Note that, as D is definite, T_D (respectively T'_D) is monotonic. We prove that $\text{ans}(D, A, L) = \text{ans}(D, A, L')$.

Let $A\theta$ be a logical consequence of $\text{comp}_{L'}(D)$. Let M be any model for $\text{comp}_L(D)$. (Such a model exists since D is definite.) By Lemma 4.6, M can be extended to a model M' for $\text{comp}_{L'}(D)$. Thus, $A\theta$ is true w.r.t. M' and hence w.r.t. M.

Conversely, let $A\theta$ be a logical consequence of $\text{comp}_L(D)$. Let M' be any model for $\text{comp}_{L'}(D)$. Let $M = T_D{\uparrow}\omega$ be the least fixpoint for T_D. Then M (together with the assignment of the identity relation to $=$) is the least model for $\text{comp}_L(D)$. Using Lemma 4.7, it is easy to prove by induction on i that $T_D{\uparrow}i = T'_D{\uparrow}i$ for all $i \geqslant 0$. As $T'_D{\uparrow}\omega$ is the least model for $\text{comp}_{L'}(D)$, $T'_D{\uparrow}\omega \subseteq M'$. As $A\theta$ is true w.r.t. M, it is also true w.r.t. M'. □

Corollary 4.9. *Every relational database is domain-independent.*

Proof. Immediate. □

The condition that D be allowed in Theorem 4.8 is necessary.

Example. Let D be the non-allowed definite database $p(x) \leftarrow$ in language L. Then, for each constant c in L, $\{x/c\}$ is in $\text{ans}(D, p(x), L)$, so $\text{ans}(D, p(x), L)$ depends on L, and D is not domain-independent.

Theorem 4.10. *Every allowed hierarchical database is domain-independent.*

Proof. Let D be an allowed hierarchical database, $L \leqslant L'$ languages extending that of D, and A an atom in L. We prove that $\text{ans}(D, A, L) = \text{ans}(D, A, L')$.

First note that every hierarchical database in language L has a unique (Herbrand) model for $\text{comp}_L(D)$. Let M (respectively M') be the unique model for $\text{comp}_L(D)$ (respectively $\text{comp}_{L'}(D)$). We prove by induction on the maximum level k of the predicates in D that $M = M'$.

(*Basis* $k=0$): As D is allowed, every clause in D is a ground unit clause. Thus, $M=\{A: A\leftarrow \text{is in } D\}=M'$.

(*Induction step*): Suppose the result holds for all allowed hierarchical databases of maximum level $k\geq 0$ and D has maximum level $k+1$. Let D_k be the set of clauses $p(t_1,\ldots,t_n)\leftarrow W$ in D such that p has level $\leq k$. Let M_k (respectively M'_k) be the unique model for $\text{comp}_L(D_k)$ (respectively $\text{comp}_L(D)_k$)). As D is hierarchical and M (respectively M') are unique, $M=T_D(M_k)$ and $M'=T'_D(M'_k)$. By the induction hypothesis, $M_k=M'_k$. Thus, by Lemma 4.7, $M=M'$. □

The condition that D be allowed in this result is again necessary.

Example. Let D be the non-allowed hierarchical database

$$p(a)\leftarrow$$
$$q(a)\leftarrow \forall x\, p(x)$$

in language L. Then $q(a)$ is a logical consequence of $\text{comp}_L(D)$ if and only if a is the only constant in L.

Theorems 4.8 and 4.10 may lead one to conjecture that every allowed database is domain-independent, but the following examples shows that the conjecture is false.

Example. Let D be the allowed database $p(a)\leftarrow \sim p(a)$. Then D is not domain-independent.

D is not domain-independent because $\text{comp}(D)$ does not have a model. Even when the completion of an allowed database does have a model, the database may not be domain-independent.

Example. Let D be the allowed database

$$t(a)\leftarrow \sim p(a)$$
$$p(a)\leftarrow \sim q(x)\wedge r(x)$$
$$q(a)\leftarrow$$
$$r(x)\leftarrow r(x)$$

in language L. Then $t(a)$ is a logical consequence of $\text{comp}_L(D)$ if and only if a is the only constant in L. Thus, D is not domain-independent.

Note that the above database is stratified [1, 2], so even the conjecture that every allowed stratified database is domain-independent is false. These examples suggest that there is no simple generalization of Theorems 4.8 and 4.10.

We now consider non-atomic queries, as discussed after the definition of domain-independent queries above. Let D be a database, L a language extending that of D, and Q a query in L. We describe the classes of databases and queries for which the set of correct answers for $\text{comp}_L(D)\cup\{Q\}$ is independent of L.

Theorem 4.11. *Let D be an allowed database, $L \leq L'$ languages extending that of D, and Q a query $\leftarrow W$ in L.*

(a) *If D is definite and W is definite, then* $\text{ans}(D, W, L) = \text{ans}(D, W, L')$.

(b) *If D is hierarchical and W is domain-independent, then* $\text{ans}(D, W, L) = \text{ans}(D, W, L')$.

Proof. (a): Let $x_1, \ldots, x_n$ be the free variables in W, 'answer' a predicate not in D or W, and D' the database $D \cup \{\text{answer}(x_1, \ldots, x_n) \leftarrow W\}$. Then D' is an allowed definite database, $\text{comp}_L(D')$ is $\text{comp}_L(D) \cup \{\forall(\text{answer}(x_1, \ldots, x_n) \leftrightarrow W)\}$, and similarly for L'. Thus, $\text{answer}(x_1, \ldots, x_n)\theta$ is a logical consequence of $\text{comp}_L(D')$ if and only if $\text{answer}(x_1, \ldots, x_n)\theta$ is a logical consequence of $\text{comp}_{L'}(D')$, and the result follows.

(b): This part follows from the observation that the unique model for $\text{comp}_L(D)$ is independent of L and the definition of a domain-independent formula. □

The restriction to definite queries in part (a) of this result cannot be relaxed to allowed queries.

Example. Let D be the definite database

$$p(x) \leftarrow p(x)$$

$$p(a) \leftarrow$$

$$q(a) \leftarrow$$

in language L and W the allowed formula $\forall x(p(x) \rightarrow q(x))$. Then W is a logical consequence of $\text{comp}_L(D)$ if and only if a is the only constant in L.

We now prove that every domain-independent database is equivalent to an allowed database in the following sense.

Theorem 4.12. *Let D be a domain-independent database and L a language extending that of D. Then there exists an allowed database D' in L such that, for any atom A in L,* $\text{ans}(D, A, L) = \text{ans}(D', A, L)$.

Proof. The following proof is due to Wallace. Let D' be the set of ground unit clauses $A \leftarrow$ such that atom A is a logical consequence of $\text{comp}_L(D)$. Then D' is clearly allowed and the result follows immediately. □

The construction of D' in the proof of Theorem 4.12 produces databases with an impractically large number of clauses. Moreover, it cannot be generalized to databases with functions. A construction similar to that used in the proof of Theorem 3.7 maintains the number of clauses in the database but can produce databases with impractically large clauses. Thus, it is again desirable to find alternative constructions that produce simpler databases.

One such construction, based on range forms, is the following. Suppose that every clause $A \leftarrow W$ in the database has the property that every variable in A is free in W and that W is range-restricted. Then, replacing each clause $A \leftarrow W$ by $A \leftarrow r(W)$ gives an equivalent allowed database, without introducing new equalities.

We have not yet been able to determine whether or not Theorem 4.12 can be generalized from atoms A to allowed formulas W. Certainly, a different construction of D' is required for such a generalization to hold.

An important reason for restricting attention to allowed databases and queries is that they have the following desirable operational properties.

Theorem 4.13. *Let D be an allowed database and Q an allowed query $\leftarrow W$. Then the following properties hold*:

(a) *Normal query evaluation of $D \cup \{Q\}$ does not flounder, i.e., it never reaches a goal containing only nonground negative literals.*

(b) *Every normal computed answer for $D \cup \{Q\}$ is a ground substitution for all free variables in W.*

Proof. The result follows immediately from Lemma 4.4 and [15, Proposition 4]. □

Non-allowed queries may not exhibit this desired operational behaviour.

Example. Consider the following query which is evaluable and range-restricted but not allowed:

$$\leftarrow \exists x \forall y (p(y) \rightarrow q(x, y)).$$

Normal query evaluation applied to this goal will reach a goal

$$\leftarrow p(y) \wedge \sim q(x, y)$$

and will flounder. The same phenomenon can occur if the database contains a clause that is evaluable but not allowed.

The completeness results given in [15] also hold for allowed databases and allowed queries as a result of Lemma 4.4.

Now, let D and Q be as in Theorem 4.13 and L a language extending that of D and Q. The examples following Theorems 4.10 and 4.11 show that the set of correct answers for $\text{comp}_L(D) \cup \{Q\}$ can depend on L. Fortunately however, the set of normal computed answers for $D \cup Q$ *is independent of* L and normal query evaluation can thus be used with confidence.

It is the combination of the simple compositional definition, the attractive declarative properties given in Theorems 4.11 and 4.12, and the desired operational properties given by Theorem 4.13 that makes the allowed databases of such importance, and preferable to databases constructed from safe or evaluable formulas.

Finally in this section, we present a theorem concerning the need for domain-closure axioms in database theory. The results arose from an attempt to understand why the domain-closure axiom is included in comp(D) for a database D [14, 19], whereas it is not included in comp(P) for a program P [3, 12]. The following theorem describes the conditions under which it is irrelevant whether the domain-closure axiom DCA is included or not.

Theorem 4.14. *Let D be an allowed database, Q a query $\leftarrow W$, L a language extending that of D and Q, and θ a ground substitution for all free variables in W.*

(a) *If D is definite and W is definite, then θ is a correct answer for* $\text{comp}_L(D) \cup \{Q\}$ *if and only if θ is a correct answer for* $\text{comp}_L(D) \backslash \{\text{DCA}\} \cup \{Q\}$.

(b) *If D is hierarchical and W is allowed, then θ is a correct answer for* $\text{comp}_L(D) \cup \{Q\}$ *if and only if θ is a correct answer for* $\text{comp}_L(D) \backslash \{\text{DCA}\} \cup \{Q\}$.

Proof. Let M' be a model for $\text{comp}_L(D) \backslash \{\text{DCA}\}$ with domain U. Then M' can be regarded as a model for $\text{comp}_{L'}(D)$, where L' is the extension of L by those elements of U not assigned by M' to constants in L. Thus, θ is a correct answer for $\text{comp}_L(D) \backslash \{\text{DCA}\} \cup \{Q\}$ if and only if θ is a correct answer for $\text{comp}_{L'}(D) \cup \{Q\}$. The results then follow from Theorem 4.11. □

The examples following Theorems 4.10 and 4.11 show that the above property does not hold for all allowed, or even all allowed stratified, databases.

5. Discussion

Our main contributions in this paper are the following. We have given a proof-theoretic characterization of domain-independent formulas. We then defined the class of allowed formulas, proved that every allowed formula is domain-independent, and showed that every domain-independent formula has an (almost) equivalent allowed formula.

We have motivated and defined the class of domain-independent databases to characterize 'reasonable' databases, defined the class of allowed databases, and proved that every allowed database that is definite or hierarchical is domain-independent and that every domain-independent database has an equivalent allowed database (in a certain sense). We indicated that allowed databases have desirable operational properties and described the conditions under which domain-closure axioms are unnecessary.

The results presented suggest that databases and queries in a deductive database system should be restricted to allowed ones. This will require the development of procedures to transform non-allowed databases and queries into simple allowed ones.

Thus, we suggest the following problems for further research. The major problem is to find larger classes of databases that are known to be domain-independent. With this in mind, note that each of the examples following Theorems 4.10 and 4.11 involve a database clause of the form $p(x) \leftarrow p(x)$. Although such a clause is a tautology, its presence changes the completed definition for p, and thus affects the set of correct answers. It can also prevent normal query evaluation from terminating. The same phenomenon can occur with more complex clauses, but it is unclear how to identify and exclude them. Thus, this problem is closely related to that of finding larger classes of databases for which normal query evaluation is complete.

The next problem is to extend our results to languages with functions. With this aim, we propose the following changes to our main definitions.

Definition. A formula W in languages L_1 and L_2 is (model-theoretically) domain-independent if, for all Herbrand interpretations I_1 for L_1 and I_2 for L_2 that assign the same relations to the predicates in W, it holds that $\text{val}(W, I_1) = \text{val}(W, I_2)$.

This appears to be the natural generalization of the model-theoretic definition of a domain-independent formula to languages with functions. It is equivalent to the current proof-theoretic definition for an appropriate definition of comp(D) [20]. The definition of an allowed formula needs to be modified only by defining

x is pos in $s = t$ if every mgu for s and t is a ground substitution for x.

The definition of a domain-independent database can remain unchanged.

A third problem is to develop practical methods of transforming domain-independent databases and queries, or large subclasses of them, into allowed databases and queries before applying normal query evaluation. The construction of range forms of range-restricted formulas as described provides a partial solution to this problem.

It is also important to investigate the operational properties of allowed databases and queries with respect to other query evaluation processes. Finally, we should consider what other properties a 'reasonable' database should have, possibly by generalizing data modelling criteria used for relational databases.

Further discussion and results concerning domain-independent databases may be found in [22].

Acknowledgment

I would like to thank Hendrik Decker, John Lloyd, Liz Sonenberg, Allen Van Gelder, and Mark Wallace for their valuable contributions to this paper. Bob Kowalski and Jean-Marie Nicolas also made useful comments on a draft. The Department of Computing, Imperial College, provided support during its preparation.

References

[1] K.R. Apt, H. Blair and A. Walker, Towards a theory of declarative knowledge, in: J. Minker, ed., *Foundations of Deductive Databases and Logic Programming* (Morgan Kaufmann, Los Altos, 1987).

[2] A.K. Chandra and D. Harel, Horn clause queries and generalizations, *J. Logic Programm.* **2**(1) (1985) 1-15.

[3] K.L. Clark, Negation as failure, in: H. Gallaire and J. Minker, eds., *Logic and Data Bases* (Plenum, New York, 1978) 293-322.

[4] E.F. Codd, Relational completeness of data base sublanguages, in: R. Rustin, ed., *Data Base Systems*, Courant Computer Science Symposium **6** (Prentice-Hall, Englewood Cliffs, NJ, 1972) 65-98.

[5] H. Decker, Integrity enforcement in deductive databases, in: *Proc. 1st Internat. Conf. on Expert Database Systems*, Charleston, SC (1987) 271-285.

[6] R. Demolombe, Utilisation du calcul des prédicats comme langage d'interrogation des bases de données, Thèse de Doctorat d'Etat, Université de Toulouse, 1982.

[7] R. Demolombe, Syntactical characterization of a subset of domain independent formulas, Tech. Rept., ONERA-CERT, Toulouse, 1982.

[8] R.A. Di Paola, The recursive unsolvability of the decision problem for the class of definite formulas, *J. ACM* **16**(2) (1969) 324-327.

[9] H. Gallaire, J. Minker and J.-M. Nicolas, Logic and databases: a deductive approach, *Comput. Surv.* **16**(2) (1984) 153-185.

[10] J.L. Kuhns, Answering questions by computer: a logical study, Tech. Rept. RM-5428-PR, Rand Corporation, Santa Monica, 1967.

[11] J.W. Lloyd, An introduction to deductive database systems, *Austral. Computer J.* **15**(2) (1983) 52-57.

[12] J.W. Lloyd, *Foundations of Logic Programming*, Symbolic Computation Series (Springer, Berlin, 1984).

[13] J.W. Lloyd and R.W. Topor, Making PROLOG more expressive, *J. Logic Programm.* **1**(3) (1984) 225-240.

[14] J.W. Lloyd and R.W. Topor, A basis for deductive database systems, *J. Logic Programm.* **2**(2) (1985) 93-109.

[15] J.W. Lloyd and R.W. Topor, A basis for deductive database systems II, *J. Logic Programm.* **3**(1) (1986) 55-67.

[16] E. Mendelson, *Introduction to Mathematical Logic* (Van Nostrand Reinhold, New York, 1979, 2nd edition).

[17] J.-M. Nicolas, Logic for improving integrity checking in relational data bases, *Acta Inform.* **18**(3) (1982) 227-253.

[18] J.-M. Nicolas and R. Demolombe, On the stability of relational queries, Tech. Rept. ONERA-CERT, Toulouse, 1983.

[19] R. Reiter, Towards a logical reconstruction of relational database theory, in: M.L. Brodie et al., eds., *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages* (Springer, Berlin, 1984) 191-233.

[20] J. C. Shepherdson, Negation as failure: a comparison of Clark's completed data base and Reiter's closed-world assumption, *J. Logic Programm.* **1**(1) (1984) 51-79.

[21] J. Shoenfield, *Mathematical Logic* (Addison-Wesley, Reading, MA, 1967).

[22] R.W. Topor and E.A. Sonenberg, On domain independent databases, in: J. Minker, ed., *Foundations of Deductive Databases and Logic Programming* (Morgan Kaufmann, Los Altos, 1987).

[23] J.D. Ullman, *Principle of Database Systems* (Pitman, London, 1982).

[24] J.D. Ullman, Implementation of logical query languages for databases, *ACM Trans. Database Systems* **10**(3) (1985) 289-321.

[25] A. Van Gelder and R.W. Topor, Safety and correct translation of relational calculus formulas, in: *Proc. 6th ACM Symp. on Principles of Database Systems*, San Diego, CA (1987) 313-327.

[26] M.Y. Vardi, The decision problem for database dependencies, *Inform. Process. Lett.* **12**(5) (1981) 251-254.