Safety and Correct Translation of Relational Calculus Formulas

Allen Van Gelder*
Stanford University

Rodney W Topor
University of Melbourne

Abstract

Not all queries in relational calculus can be answered "sensibly" once disjunction, negation, and universal quantification are allowed The class of relational calculus queries, or formulas, that have "sensible" answers is called the *domain independent* class, which is known to be undecidable Subsequent research has focused on identifying large decidable subclasses of domain independent formulas In this paper we investigate the properties of two such classes the *evaluable* formulas and the *allowed* formulas Although both classes have been defined before, we give simplified definitions, present short proofs of their main properties, and describe a method to incorporate equality

Although evaluable queries have sensible answers, it is not straightforward to compute them efficiently or correctly We introduce *relational algebra normal form* for formulas from which form the correct translation into relational algebra is trivial We give algorithms to transform an evaluable formula into an equivalent *allowed* formula, and from there into relational algebra normal form Our algorithms avoid use of the so-called *Dom* relation, consisting of all constants appearing in the database or the query

Finally, we describe a restriction under which every domain independent formula is evaluable, and argue that evaluable formulas may be the largest decidable subclass of the domain independent formulas that can be efficiently recognized

*Supported by NSF grant IST-84-12791 and a grant of IBM Corp

1 Introduction

With the increased interest in development of deductive database systems and integration of logic programming languages such as Prolog with relational database systems, it has become more important that relational query systems be able to handle a wider range of relational calculus formulas correctly and efficiently In particular, disjunction, negation, and universal quantification over subformulas, which are excluded from the class of *conjunctive queries* [Ull80], should be available Current "industrial strength" implementations handle the class of conjunctive queries well, but leave much to be desired in the areas mentioned, we shall give an example later In defense of these implementations, we should point out that the large majority of queries posed by typical users to traditional databases fall into the class of conjunctive queries However, in sophisticated systems of the future we envision the queries often being generated not by the user typing them in at the terminal, but by a layer of software positioned between the user and the relational database system This software will access a large set of deductive rules in addition to the user's query in order to construct relational calculus formulas The Nail! project at Stanford University [MUVG86] is just one example of several research projects headed in this direction

2 Problem Statement and Background

In this paper we shall be concerned with two main questions

1 Which relational calculus queries can be answered sensibly?
2 How can such queries be answered?

For our purposes, answering a query means evaluating a relational calculus formula By "sensible" we mean that values in any logically correct answer are limited to values that appear in the query itself or in database relations mentioned in the query

Not all queries in relational calculus can be answered sensibly Two simple examples that cannot be answered sensibly are

$$F(x) \stackrel{\mathrm{def}}{=} \neg P(x)$$
$$G(x,y) \stackrel{\mathrm{def}}{=} P(x) \vee Q(y)$$

where P and Q are database relations $F(x)$ holds for arbitrary x's that are not in the database, and $G(x,y)$ holds for arbitrary y values when $P(x)$ is true, and *vice versa*

In the following section, we describe previous attempts to characterize those classes of queries that can be answered sensibly

Evaluation of relational calculus queries can be performed either by translation into a set of clauses suitable for a Prolog interpreter [LT84, Top86, Dec86], or by translation into a relational algebra expression Here, we are concerned solely with the second approach

Translation of a relational calculus query that includes disjunction and/or negation is a theoretically solved problem [Ull80], provided the query is "safe" However, the practical difficulties are such that several commercial database query systems give intuitively unexpected results on such queries

Here is a "real life" example Essentially, a user posed the query (we simplify the syntax)

select	$R1$ *name*
from	$R1$, $R2$, $R3$
where	$R1$ *name* $=$ $R2$ *name*
or	$R1$ *name* $=$ $R3$ *name*,

and was quite surprised to find out that the answer was nil when relation $R3$ was empty, even though there were matches between $R1$ and $R2$ This user was even more surprised when the vendor claimed that this behavior was correct! In fact, the semantics of QUEL [Ull80] *do* support this behavior, and several systems whose query language is an outgrowth of QUEL give nil answers

While the vendors are saved by the "fine print," which says that even though their language ***looks*** like relational calculus, it is really a relational algebra expression in disguise, the situation is hardly satisfactory from the user's point of view The QUEL interpretation has only been proven to yield correct translations of *conjunctive* relational calculus queries (defined below) [Ull80] The problems of correct translation of more general relational calculus formulas still need to be addressed

2 1 What are the Problems?

Conjunctive query formulas are those that use only $\exists$ and $\wedge$ (Equality can be represented in conjunctive queries by repetition of variables and substitution of constants, for simplicity, we do not consider "built-in" predicates such as $<$, $>$, etc) The translation of such a formula into an equivalent relational algebra expression is straightforward and well-known Informally, $A(u,v,w,xy) \wedge B(u,v,y,z)$ becomes an equijoin on the columns of u and v, and $\exists x A(x,y,z)$ becomes a projection that eliminates the column for x Essentially, all such formulas can be translated

The situation changes when we introduce disjunction and/or negation We intend to handle disjunction algebraically by ***union*** and handle negation by ***set difference*** For example, $P(x,y) \vee Q(x,y)$ can be evaluated by $P \cup Q$, and $P(x,y) \wedge \neg Q(x)$ can be evaluated by P **diff** Q More generally, to have a simple representation in relational algebra, both operands of "$\vee$" must have the same variables, while negations must appear in the form $A \wedge \neg B$ where B's variables are a subset of A's [Ull80]

These limitations give rise to ill-behaved cases as demonstrated by the two earlier examples

$$F(x) \stackrel{\mathrm{def}}{=} \neg P(x)$$
$$G(x,y) \stackrel{\mathrm{def}}{=} P(x) \vee Q(y)$$

The two problems here, which are the main problems aside from handling equality, are

- The terms of a disjunction do not have the same set of free variables
- A variable in a negative atom is not limited in its range by positive atoms elsewhere in the formula

Once we develop tools to handle these problems, then universal quantifiers will not present any new problems, we will be able to rewrite $\forall x$ as $\neg\exists x\neg$ at the appropriate moment

The situation is really more complicated than it might appear at first glance, because the problem in a subformula can often be cured by some other part of the overall formula Thus even though the query

$$G(x,y) \stackrel{\mathrm{def}}{=} P(x) \vee Q(x,y)$$

is definitely not "reasonable," because it holds for arbitrary y values when $P(x)$ is true, nevertheless, the query

$$F(x) \stackrel{\mathrm{def}}{=} \exists y G(x,y) \equiv \exists y (P(x) \vee Q(x,y))$$

may well be considered reasonable The naive translation into $\pi_1(P \cup Q)$, where π_1 means "project onto column 1," presents problems because the operation $P \cup Q$ makes no sense However, in this case $F(x)$ has an equivalent form,

$$F(x) = (P(x) \vee \exists y Q(x, y))$$

for which the naive translation is correct, and is $P \cup \pi_1(Q)$

Our goal is develop a systematic method to distinguish the curable problems, such as the above, from the uncurable ones, such as $\exists y(P(x) \vee Q(y))$, and to provide correct transformations for the curable ones

2 2 Previous Work

There have been several attempts to define a "reasonable" class of queries, i e , a class with the following desirable properties

- The constants in the database and the query provide a sufficient domain for the values in the answer Formulas with this property are called *domain independent* [Fag80, Mak81]
- There is an efficient way to decide if the query formula is "reasonable" and if so, to translate the relational calculus formula into a relational algebra expression whose evaluation gives the correct answer
- There is an efficient way to evaluate the resulting relational algebra expression

The class of conjunctive queries has these properties, as shown in [Ull80], but this class is rather limited The class of *domain independent* formulas [Fag80, Mak81], which by its definition is the largest class having the first property listed above, represents a generalization of *safe* formulas, introduced in [Ull80] However, the domain independent class has been shown in [ND82] to be equivalent to the class of *definite* formulas defined in [Kuh67], and definite formulas were shown to be not recursive in [DiP69]

Other researchers have subsequently proposed decidable subclasses of domain independent formulas, including *range restricted* formulas [Nic82, Dec86], *evaluable* formulas [Dem82], and *allowed* formulas [Top86] We give their definitions later, as we discuss them

Of these, the evaluable formulas comprise the largest class, but the definition of this class in [Dem82] occupies three pages, its complex definition makes it unwieldy to work with, as evidenced by the fact that it required ten pages just to prove that it is a subclass of domain independent formulas, moreover, there is no attempt there to describe how to actually evaluate evaluable formulas, i e , how to translate them correctly into relational algebra expressions

The allowed formulas, although a strict subclass of the evaluable formulas, are the easiest (among the above-mentioned classes) to translate into relational algebra

The range restricted formulas comprise the evaluable formulas that are in disjunctive normal form or conjunctive normal form [Dem82] In an important step toward practical evaluation, Decker [Dec86] has shown how to transform any range restricted formula into an equivalent[1] *range form* that is suitable for Prolog-style "tuple at a time" evaluation

3 Summary of Results

In this paper we give a much simpler definition of evaluable formulas With this simpler definition, it is more feasible to prove properties of the evaluable class, and to see the relationship between allowed formulas and evaluable formulas We show that the evaluable class is invariant under a set well-known equivalences that can be used as rewrite rules (e g , DeMorgan's laws), which we call *conservative* transformations This invariance makes it easy to see that every evaluable formula can be conservatively rewritten in prenex-literal normal form (Def 4 1) However, the *evaluable* property is not always preserved under distribution of $\wedge$ over $\vee$ or $\vee$ over $\wedge$ Using distribution is apparently a necessary step to put certain formulas into an equivalent form that can be "transliterated" into relational algebra This is our motivation for transforming evaluable formulas into allowed formulas, which *are* invariant under distribution

One of our main results is an algorithm that transforms any evaluable formula into an equivalent allowed formula

Another main result is that every allowed formula can be effectively translated correctly into a relational algebra expression

At this point we should mention two properties of formula transformations (either into other formulas or into relational algebra expressions) that we consider unacceptable, and wish to avoid The first property is that the transformation does not necessarily produce a logically equivalent formula, but is only guaranteed to do so if the input formula is a certain class (such as the domain independent class) This puts the burden on the user of providing correct input, or getting erroneous results with no warning The second unacceptable method is to explicitly form the so-called *Dom* relation, consisting of all constants present in the database and the query Both these drawbacks are present, for example, in the rewrite

[1] By *equivalent* we shall always mean *logically equivalent*

rule

$$\neg P(x,y) \equiv Dom(x) \wedge Dom(y) \wedge \neg P(x,y) \longrightarrow Dom \times Dom - P$$

Both of our transformation algorithms have the attractive property that such tactics are not required

Finally, we shall show that the class of evaluable formulas is the largest practical subclass of domain independent formulas in a certain sense Essentially, the domain independent class is not recursive because a given formula may have a subformula that is superficially not domain independent, but is unsatisfiable, hence is actually domain independent (vacuously) [2] However, formulas in which no predicate symbol is repeated cannot possibly have unsatisfiable subformulas We show that formulas in this class are evaluable if and only if they are domain independent, and discuss the ramifications

4 Notation and Definitions

We assume the reader is familiar with the standard notation and terminology of logic, relational calculus, and relational algebra [Man74, Ull80] We shall abbreviate "first order well formed formula" to *formula*, and "atomic formula" to *atom* A *literal* is either an atom or a negated atom We assume the absence of function symbols (other than constants) throughout We shall use P and Q to denote predicate symbols or atoms that correspond to a database relation, we call these *edb* predicates We use A, B, to denote formulas and subformulas, we use a, , d as constants, u, , z as variables, and s and t to represent a term that may be either a variable or a constant

We adopt a sort of vector notation $\vec{x}$ to denote a tuple $(x_1, \quad, x_n)$, where n may be zero Thus the notation $A(x, \vec{y})$ denotes a formula in which x is a free variable and there are zero or more other free variables y_i that are of interest, in addition, A may contain still other free variables that are not currently of interest

In a similar vein, we write $\forall \vec{x}$ for $\forall x_1 \quad \forall x_n$, and write $\exists \vec{x}$ for $\exists x_1 \quad \exists x_n$ We also use "%" as a "quantifier variable," standing for either $\forall$ or $\exists$, or in the case of $\% \vec{x}$, for a specific *string* of (possibly mixed) quantifiers We assume that no quantified variable occurs outside the scope of its quantifier, i e, we avoid $(\exists x A(x) \wedge \exists x B(x))$ and use instead $(\exists x_1 A(x_1) \wedge \exists x_2 B(x_2))$

We shall use $\equiv$ to denote logical equivalence and $\Rightarrow$ to denote logical implication, both denote relations between formulas, not symbols within formulas In addition, $\stackrel{\text{def}}{=}$ is often used to mean "is defined as" to give names to formulas We occasionally use "[ ]" as synonyms for "( )" for readability

We adopt the usual definitions ([Man74], etc) for *prenex normal form*, *conjunctive normal form*, and *disjunctive normal form*, which we abbreviate to PNF, CNF and DNF, respectively We shall also introduce *relational algebra normal form*, abbreviated RANF (See Def 9 2) In addition, we shall have several occasions to refer to the following normal form

Definition 4.1 A formula is said to be in *prenex-literal normal form* (PLNF) if it is in PNF and all negations are immediately above the atoms (This is sometimes called *negative normal form*) □

As usual in the context of normal forms, we regard $\wedge$ and $\vee$ as polyadic operators taking zero or more operands, with zero operands, $\wedge() \equiv$ *true* and $\vee() \equiv$ *false* A *clause* is a conjunction of literals or a disjunction of literals

5 Evaluable and Allowed Classes of Formulas

In this section we define the classes of *evaluable* formulas and *allowed* formulas, and give some of their properties The term *evaluable* is due to R Demolombe [Dem82] We use the same term because the class is the same, although our definition is different Actually, there is a minor difference in that we treat $x = c$, where c is a constant, as though it were $x \stackrel{a}{=} c$, where $\stackrel{a}{=}$ is an *edb* predicate, in effect, this case is not mentioned in [Dem82], but could be incorporated easily

5 1 The gen and con Relations

To define *evaluable* and *allowed* we first need to define certain relations between variables and (sub)formulas We have chosen the names *gen* and *con* for these key relations They are abbreviations for *generated* and *consistent* Our relation *generated* is called *restricted* in [Dem82] and *pos* in Top86, to avoid taking sides we have chosen a third name Also, our *consistent* is similar to, but not quite the same as, what [Dem82] calls *positive* We prefer to use the terms *positive* and *negative* to describe the polarity of atoms or subformulas within a formula As mentioned before, a subformula is considered to be *positive* if it falls under an even number of negations, and *negative* if it falls under an odd number

Definition 5 1 The essentials of the definitions for *gen* and *con* are presented in Fig 1 in a rule format

[2] The situation is not this simple, but this is the central idea

$$
\begin{array}{lll}
gen(x,\,P) & \textbf{if} & edb(P)\ \&\ free(x,\,P) \\
gen(x,\,x=c) & \textbf{if} & constant(c) \\
 & & \\
gen(x,\,\neg A) & \textbf{if} & pushnot(\neg A,\,B)\ \&\ gen(x,\,B) \\
gen(x,\,\exists yA) & \textbf{if} & distinct(x,\,y)\ \&\ gen(x,\,A) \\
gen(x,\,\forall yA) & \textbf{if} & distinct(x,\,y)\ \&\ gen(x,\,A) \\
gen(x,\,A \vee B) & \textbf{if} & gen(x,\,A)\ \&\ gen(x,\,B) \\
gen(x,\,A \wedge B) & \textbf{if} & gen(x,\,A) \\
gen(x,\,A \wedge B) & \textbf{if} & gen(x,\,B)
\end{array}
$$

$$
\begin{array}{lll}
con(x,\,P) & \textbf{if} & edb(P)\ \&\ free(x,\,P) \\
con(x,\,x=c) & \textbf{if} & constant(c) \\
con(x,\,A) & \textbf{if} & \textbf{not}\, free(x,\,A) \\
con(x,\,\neg A) & \textbf{if} & pushnot(\neg A,\,B)\ \&\ con(x,\,B) \\
con(x,\,\exists yA) & \textbf{if} & distinct(x,\,y)\ \&\ con(x,\,A) \\
con(x,\,\forall yA) & \textbf{if} & distinct(x,\,y)\ \&\ con(x,\,A) \\
con(x,\,A \vee B) & \textbf{if} & con(x,\,A)\ \&\ con(x,\,B) \\
con(x,\,A \wedge B) & \textbf{if} & gen(x,\,A) \\
con(x,\,A \wedge B) & \textbf{if} & gen(x,\,B) \\
con(x,\,A \wedge B) & \textbf{if} & con(x,\,A)\ \&\ con(x,\,B)
\end{array}
$$

Figure 1 Definitions by rules of *gen* and *con*

similar to a Prolog program[3] We intend that the relations *gen* and *con* hold only when they can be established by a finite number of applications of these rules □

Read the & 's that separate subgoals (to the right of the "**if**") as "and" For example, the first rule reads, "x is generated in P if P is an *edb* atom, and x is free in P"

Several predicates appear in these rules to support the definitions of *gen* and *con* We intend that they be interpreted as follows

- $edb(P)$ holds precisely when P is an atom whose predicate symbol represents a database relation
- $free(x,\,A)$ holds when variable x occurs freely in formula A
- $distinct(x,\,y)$ holds when x and y are different variables
- $constant(c)$ holds when c is a constant
- *pushnot* rewrites its first argument into an equivalent formula without "¬" at the top, by applying DeMorgan's laws, changing $\neg\exists$ to $\forall\neg$, or changing $\neg\forall$ to $\exists\neg$, it fails when this is impossible, i e , when A is an atom The second argument becomes the transformed formula when *pushnot* succeeds

Intuitively, $gen(x,\,A)$ means that A can generate all the needed values of x, as though it were a database relation In other words, A holds for only a finite set of values of x (assuming finite *edb* relations, of course)

Lemma 5.1 For every variable x and formula A, $gen(x,\,A)$ implies $con(x,\,A)$

Proof Use structural induction on the subformulas of A ▮

[3] Prolog *cognoscenti* are warned not to take the syntax too seriously, x and y are still to be interpreted as variables

Example 5 1 The converse to Lemma 5 1 is false In the following, $con(x,\,A)$ holds but $gen(x,\,A)$ does not hold

$$A \stackrel{\text{def}}{=} P(x,y) \vee Q(y)$$
$$A \stackrel{\text{def}}{=} \neg Q(y)$$

Note that x need not appear in A □

Intuitively, $con(x,\,A)$ means that for any assignment to other variables of A, say $\vec{y} = \vec{y}_0$, either

- A can generate all the needed values of x, or
- $A(x, \vec{y}_0)$ holds for no x, or
- $A(x, \vec{y}_0)$ holds for all x

Figure 2 shows a geometric interpretation of *con* If *con* holds for all the free variables of A and the underlying *edb* relations are finite, then the set of points where A holds can be represented as a finite collection of points, lines, planes, and hyperplanes

Also, from a logic programming viewpoint, we can think of A as a goal that may succeed without instantiating all of its arguments

5.2 Evaluable and Allowed Formulas

Definition 5 2 A formula F is *evaluable* or has the *evaluable property* if and only if

- For every variable x that is free in F, $gen(x,\,F)$ holds
- For every subformula of the form $\exists xA$, $con(x,\,A)$ holds
- For every subformula of the form $\forall xA$, $con(x,\,\neg A)$ holds

□

Definition 5.3: A formula F is *allowed*, or has the *allowed property* if and only if

- For every variable x that is free in F, $gen(x,\,F)$ holds
- For every subformula of the form $\exists xA$, $gen(x,\,A)$ holds

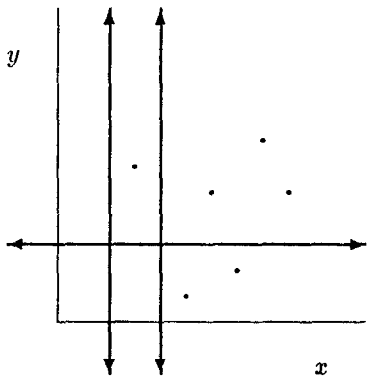

Figure 2 Geometric interpretation of the *con* property for $A(x,y) \stackrel{\text{def}}{=} P(x) \vee Q(y) \vee R(x,y)$

- For every subformula of the form $\forall x A$, $gen(x, \neg A)$ holds

$\square$

Rather than prove that our definition of evaluable yields the same class as [Dem82], it is easier to just reprove the important properties of the class We shall show that every evaluable formula (and hence every allowed formula) is domain independent in Section 10, after developing some more machinery

Theorem 5 2 Every allowed formula is evaluable

Proof Immediate from Lemma 5 1 ∎

Example 5 2 The converse of Theorem 5 2 is false The following formulas are evaluable but not allowed

$$F(y) \stackrel{\text{def}}{=} \exists x[(P(x,y) \vee Q(y)) \wedge \neg R(y)]$$
$$G \stackrel{\text{def}}{=} \exists y \forall x(\neg P(x) \vee S(y,x))$$

With appropriate interpretations of P and S formula G corresponds to the question, "Does some supplier supply all parts?"

Also, note that removing the outer quantifier makes both F and G not evaluable The problem with the apparently harmless variant, "What suppliers supply all parts?" is that if $P(x)$ is empty, then G holds for arbitrary y $\square$

5 3 Equality in Evaluable Formulas

The definition of evaluable in this section adopts a "middle of the road" approach to equality It is quite conservative with respect to equality between two variables, since $gen(x, x = y)$ and $con(x, x = y)$ never hold Formulas satisfying Def 5 2 may be said to be *strict sense evaluable* In Appendix A we describe transformations that remove many instances of such equalities, and yield an "equality reduced" form We call formulas that can be transformed into evaluable formulas by means of these transformations *wide sense evaluable*

On the other hand, defining $gen(x, x = c)$ to hold involves going beyond strict relational calculus as defined in [Ull80], in that it allows "disembodied" variables into a formula that do not appear in any *edb* atoms One way to justify this is to assume that the underlying query answering system will (in effect) form a relation on the fly, call it $\stackrel{q}{=}$, containing tuples (c_i, c_i) for the constants c_i that appear in the query Then the system treats $x = c$ as though it were $x \stackrel{q}{=} c$, an *edb* atom It is easy to adapt our methods to systems that lack this capability Simply remove the rules for $gen(x, x = c)$ and $con(x, x = c)$ in Figs 1 and 5 and treat $x = c$ like $x = y$ throughout

Allowing $x = c$ is the only way to have values in the answer that were not in the database Such values might serve as defaults For example, if P represents *part* and S represents *supplies*, then

$$P(x) \wedge (S(y,x) \vee (\forall z \neg S(z,x) \wedge y = \text{none}))$$

appears to be a plausible query that a system should handle

6 Conservative and Distributive Transformations of Formulas

In this section we study the effects of various logical transformations on the evaluable and allowed properties of formulas, with a view to identifying sets of transformations under which these properties are invariant

Figure 3 shows some standard equivalences that are frequently useful to manipulate formulas [Ack68, Man74] Note that they preserve the number of atoms, and hence preserve the number of binary logical operators We show that the evaluable property is invariant under transformations based on these identities

Definition 6 1 We say that G is a *conservative transformation* of F if G can be obtained by replacing a subformula of F according to one of the equivalences in Fig 3, or by a series of such replacements $\square$

Lemma 6 1 The relations *gen* and *con* defined in Fig 1 are invariant under conservative transformations ((E1–10) of Fig 3) That is, if $G(y)$ is a conservative transformation of $F(y)$, then $gen(y, F(y)) \Leftrightarrow gen(y, G(y))$, and similarly for *con*

$$
\begin{array}{rcll}
\neg\neg A & \equiv & A & \textbf{(E1)} \\
\neg(A \wedge B) & \equiv & \neg A \vee \neg B & \textbf{(E2)} \\
\neg(A \vee B) & \equiv & \neg A \wedge \neg B & \textbf{(E3)} \\
\neg\forall x A(x)) & \equiv & \exists x \neg A(x) & \textbf{(E4)} \\
\neg\exists x A(x)) & \equiv & \forall x \neg A(x) & \textbf{(E5)} \\
\%x A(x, \vec{y}) & \equiv & \%v A(v, \vec{y}) & \textbf{(E6)} \\
\forall x(A(x) \vee B) & \equiv & \forall x A(x) \vee B & \textbf{(E7)} \\
\exists x(A(x) \wedge B) & \equiv & \exists x A(x) \wedge B & \textbf{(E8)} \\
\exists x(A(x) \vee B(x)) & \equiv & \exists x_1 A(x_1) \vee \exists x_2 B(x_2) & \textbf{(E9)} \\
\forall x(A(x) \wedge B(x)) & \equiv & \forall x_1 A(x_1) \wedge \forall x_2 B(x_2) & \textbf{(E10)}
\end{array}
$$

Figure 3 The equivalences upon which *conservative transformations* are based "%" stands for $\exists$ or $\forall$

$$
\begin{array}{rcll}
A \wedge (B \vee C) & \equiv & (A \wedge B) \vee (A \wedge C) & \textbf{(E11)} \\
A \vee (B \wedge C) & \equiv & (A \vee B) \wedge (A \vee C) & \textbf{(E12)} \\
\exists x(x = y \wedge A(x, y)) & \equiv & A(y, y) & \textbf{(E13)} \\
\forall x(x \neq y \vee A(x, y)) & \equiv & A(y, y) & \textbf{(E14)}
\end{array}
$$

Figure 4 Other useful equivalences distributive laws and equality elimination We use $x \neq y$ to abbreviate $\neg x = y$

Proof This is merely a matter of applying the definitions For example, suppose (E10) applies, i e ,

$$
\begin{aligned}
F(x, y) &\stackrel{\text{def}}{=} \forall x(A(x, y) \wedge B(x, y)) \\
G(x, y) &\stackrel{\text{def}}{=} \forall x_1 A(x_1, y) \wedge \forall x_2 B(x_2, y)
\end{aligned}
$$

(y may be absent from A or B) If $con(y,\ F(x, y))$ holds, then $con(y,\ A(x, y) \wedge B(x, y))$ also holds, and at least one of the following three is true

- $gen(y,\ A(x, y))$ holds Then $gen(y,\ \forall x_1 A(x_1, y))$ also holds
- $gen(y,\ B(x, y))$ holds Then $gen(y,\ \forall x_2 B(x_2, y))$ also holds
- Both $con(y,\ A(x, y))$ and $con(y,\ B(x, y))$ hold Then $con(y,\ \forall x_1 A(x_1, y))$ and $con(y,\ \forall x_2 B(x_2, y))$ also hold

And so $con(y,\ G(x, y))$ is seen to hold The other direction and other cases are similar ▮

Theorem 6 2 If A is evaluable and B is a conservative transformation of A, then B is evaluable

Proof (Sketch) The only cases not handled by Lemma 6 1 involve moving the quantifier for the first argument of a *con* by means of (E7-10) ▮

Corollary 6 3 Every evaluable formula can be conservatively transformed into an equivalent evaluable formula in PLNF (Def 4 1)

Corollary 6 4 Every evaluable formula can be conservatively transformed into an equivalent evaluable formula that contains no universal quantifiers and has negations only immediately above atoms and existential quantiers

Example 6.1 The *allowed* property may not be preserved by the conservative transformations (E7-8) Thus, allowed formulas do not always have a conservative transformation into prenex normal form E g , the allowed formula

$$\exists x A(x) \vee B$$

can be conservatively transformed to

$$\exists x(A(x) \vee B)$$

which is not allowed □

Although the distributive laws, shown in Fig 4, cannot be applied indiscriminately, some properties are preserved in some cases, as described in the next lemma

Lemma 6 5 The relation *con* defined in Fig 1 is invariant under (E11) of Fig 4 ("pushing ands) That is,

$$con(x,\ A \wedge (B \vee C))$$

if and only if

$$con(x,\ (A \wedge B) \vee (A \wedge C))$$

In addition, *gen* is invariant under both distributive laws (E11-12) of Fig 4

Proof (Sketch) Case analysis, using the definitions ▮

Example 6.2. As pointed out in [Dem82], "pushing ors" (E12) does not always preserve *con* For example, consider

$$
\begin{aligned}
F &\stackrel{\text{def}}{=} P(x) \vee (Q(x, y) \wedge \neg R(y)) \\
G &\stackrel{\text{def}}{=} (P(x) \vee Q(x, y)) \wedge (P(x) \vee \neg R(y))
\end{aligned}
$$

Here $con(y,\ F)$ holds, but $con(y,\ G)$ fails □

6.1 Invariance of Allowed Formulas under Distribution

In Section 8 we describe an algorithm to transform an evaluable formula into an equivalent allowed formula One motivation for this transformation is that the *allowed* property is preserved by the distributive laws, whereas the *evaluable* property is not The final translation into relational algebra normal form (Section 9) frequently requires application of the distributive laws

Theorem 6 6. If A is allowed and B is obtained from A by either

- a distributive law transformation (E11-12) of Fig 4, or
- a conservative transformation except for (E7-8),

then B is also allowed

Proof. The distributive laws are immediate from Lemma 6 5 The rest is similar to Theorem 6 2, except that we need to check that the needed *gen* relations are present when (E9-10) are used

Example 6 3 The following formula shows that "pushing ands" (E11) does not always preserve the evaluable property Let $F(z) \stackrel{\text{def}}{=} \forall x \exists y A(x,y,z)$, where

$$A(x,y,z) \stackrel{\text{def}}{=} R(y,z) \wedge (Q(x) \vee \neg P(x))$$

Since

$$\neg A(x,y,z) \equiv \neg R(y,z) \vee (\neg Q(x) \wedge P(x))$$

we have $con(x, \neg A)$, as required for F to be evaluable

Pushing the "and" gives

$$B(x,y,z) \stackrel{\text{def}}{=} (R(y,z) \wedge Q(x)) \vee (R(y,z) \wedge \neg P(x))$$

and the corresponding $G \stackrel{\text{def}}{=} \forall x \exists y B(x,y,z)$ However, $con(x, \neg B)$ does not hold, so G is not evaluable The problem is that "pushing and" in A is the same as "pushing or" (E12) in $\neg A$ This is the one distributive transformation that may not preserve *con* □

7 Range Restricted Formulas

Range restricted formulas are based on disjunctive and conjunctive normal forms, and represent one of the first decidable subclasses of domain independent formulas to be studied [Nic82] Putting formulas into normal forms requires the use of distributive laws (E11-12) of Fig 4 Since the distributive laws do not always preserve the evaluable property, it is not too surprising that certain evaluable formulas become non-evaluable if we simply put them into DNF in an attempt to make an equivalent range restricted formula, as shown by Example 6 3 However, we show that every evaluable formula (and only those) has an associated ***pair*** of formulas in DNF and CNF that satisfy conditions quite similar those required for range restricted formulas This theorem provides an alternate recognition mechanism for evaluable formulas

Definition 7 1 Let $F \stackrel{\text{def}}{=} \%\vec{x}M$ be a formula in disjunctive normal form, where $M \stackrel{\text{def}}{=} (D_1 \vee \cdots \vee D_n)$ Let $M' \stackrel{\text{def}}{=} (C_1 \vee \cdots \vee C_m)$ be the conjunctive normal form of M constructed by applying the distributive law (E12) of Fig 4 Then F is *range restricted* if the following properties hold

1 For every free variable x in F, x occurs in a positive atom in every D_i, i e , $gen(x, M)$ holds
2 For every existentially quantified variable x in F, x occurs in a positive atom in every D_j in which x occurs, i e , $con(x, M)$ holds
3 For every universally quantified variable x in F x occurs in a negative atom in every C_k in which x occurs, i e , $con(x, \neg M')$ holds

□

Item 3 in the above definition was stated somewhat differently in [Dem82]

3′ For every universally quantified variable x in F, if x occurs in any positive atom, then there is some clause D_j such that every atom of D_j is negative and contains x (Either $con(x, \neg D_i)$ holds for all D_i or $gen(x, \neg D_j)$ holds for some D_j, i e $con(x, \neg M)$ holds)

The equivalence of the two definitions follows from Lemma 6 5, since $\neg M'$ is obtained from $\neg M$ by pushing and's (E11)

Theorem 7 1 (Demolombe [Dem82]) Let F be a formula in disjunctive normal form Then F is evaluable if and only if F is range restricted

Proof Immediate from the definition, Lemma 6 1, and Lemma 6 5 ▮

Demolombe observes that a similar result holds for formulas in conjunctive normal form

This theorem can be generalized to apply to all evaluable formulas

Definition 7 2· Let $cnf(F)$ (resp , $dnf(F)$) be the conjunctive (resp , disjunctive) normal form of formula F constructed by applying conservative transformations and distributive law (E11) (resp (E12))
□

Theorem 7 2 Let F be a formula with

$$dnf(F) \stackrel{\text{def}}{=} \%\vec{x}M_d \stackrel{\text{def}}{=} \%\vec{x}(D_1 \vee \cdots \vee D_n)$$
$$cnf(F) \stackrel{\text{def}}{=} \%\vec{x}M_c \stackrel{\text{def}}{=} \%\vec{x}(C_1 \wedge \cdots \wedge C_m)$$

Then F is evaluable if and only if the following properties hold

1 For every free variable x in F, x occurs in a positive atom in every D_i, i e , $gen(x, M_d)$ holds
2 For every existentially quantified variable x in $dnf(F)$, x occurs in a positive atom in every D_j in which x occurs, i e , $con(x, M_d)$ hold

3. For every universally quantified variable x in $cnf(F)$, x occurs in a negative atom in every C_k in which x occurs, i.e., $con(x, \neg M_c)$ holds

Proof (Sketch) Theorem 6.2 and Lemma 6.5 allow us to put F into prenex-literal normal form (Def 4.1) and push and's in M, while preserving *gen* and *con*. Pushing or's in M is the dual of pushing and's in $\neg M$ ∎

Again we remark that $dnf(F)$ and $cnf(F)$ may not themselves be evaluable, as shown in Example 6.3

8 Transformation into an Allowed Formula

We now describe a procedure to transform any evaluable formula into an equivalent allowed formula. The approach used in [Dec86] to convert a range-restricted formula into "range form," which is nearly the same as "allowed," can be generalized quite nicely with the aid of the rules for *gen* and *con* in Fig 1.

The basic idea is to add a third argument G to *gen* and *con*, which functions as a "generator" of sorts. The modified rules are shown in Fig 5. $G(x)$ will be a disjunction of certain atoms in A, either *edb* or of the form $x = c$. (Both A and G may contain other variables besides x.) We see that the G in the conclusion, or head, of each rule is inherited naturally from the subgoals. The G in *con* is similar, except we need to provide for the possibility that x does not even occur in A. For this, we introduce "$\perp$" as a placeholder, it may be thought of as a one place *edb* predicate whose relation is always empty

Definition 8.1 For any formula G, not necessarily containing x and possibly containing other free variables, $\exists{*}G(x)$ denotes G with all variables except x existentially quantified, except that $\exists{*}\perp$ denotes *false* □

Definition 8.2 The operation of *truth value simplification* consists of applying the following simplifications to a formula as long as possible

$$\begin{array}{rr} \neg false \rightarrow true & \neg true \rightarrow false \\ A \wedge false \rightarrow false & A \wedge true \rightarrow A \\ A \vee false \rightarrow A & A \vee true \rightarrow true \\ \%x\, false \rightarrow false & \%x\, true \rightarrow true \end{array}$$

□

The following lemma partly motivates the definition of the third arguments of *gen* and *con*

Lemma 8.1 Let *gen* be defined as in Fig 5. Let x be any variable and A and G be any formulas such that $gen(x, A, G)$ holds. Then

$$\exists * A(x) \Rightarrow \exists * G(x)$$

In other words, in any interpretation the set of values of x for which $A(x)$ holds is a subset of those for which $G(x)$ holds

Proof: Straightforward by structural induction, observing that $\forall y A \Rightarrow \exists y A$ ∎

In the following algorithm $genify(F)$ we describe the local transformation that, when repeatedly applied, makes an evaluable formula into an allowed formula with respect to all of its bound variables. Beforehand, we check that $gen(x, F)$ holds for each free variable x, and replace $\forall y$ by $\neg\exists y \neg$ throughout

Algorithm 8.1: genify(F)

INPUT A formula F with no universal quantifiers such that $gen(x, F)$ holds for all free variables x in F

OUTPUT An allowed formula equivalent to F, or a message that F is not evaluable

PROCEDURE

1. Let F be of the form $\exists x A$, where x may not appear in A and A may have other variables as well
   (a) If $gen(x, A(x), G(x))$ holds, there is nothing to do here, set $F_1 \stackrel{\text{def}}{=} F$ and continue at (3)
   (b) If $con(x, A(x), G(x))$ does *not* hold, then F is not evaluable. Issue an error message and halt
   (c) If x is not free in A (detected by $G = \perp$), then set $F_1 \stackrel{\text{def}}{=} A$ and continue at (3)
   (d) If $con(x, A(x), G(x))$ holds (but *gen* does not). Recall that G is a disjunction $P_1 \vee \dots \vee P_k$ of atoms that appear in A. Let R be the new formula that results from replacing each occurrence of $P_1, \dots, P_k$ in A by *false*, and carrying out truth value simplifications [4] Set

   $$F_1 \stackrel{\text{def}}{=} \exists x(\exists{*}G(x) \wedge A(x)) \vee R$$

   and continue at (3)
2. If F is not of the form $\exists x A$, set $F_1 \stackrel{\text{def}}{=} F$ and continue at (3)
3. If F_1 is an atom, return F_1, otherwise, recursively call *genify* on each principal subformula of F_1, and return the combined results. That is, if $F_1 \stackrel{\text{def}}{=} A \vee B$, then return $genify(A) \vee genify(B)$, etc

[4] Quantified variables in A are given new names in R, of course

$$
\begin{array}{ll}
gen(x,\ P,\ P) & \textbf{if}\ edb(P)\ \&\ free(x,\ P) \\
gen(x,\ x=c,\ x=c) & \textbf{if}\ constant(c) \\
 & \\
gen(x,\ \neg A,\ G) & \textbf{if}\ pushnot(\neg A,\ B)\ \&\ gen(x,\ B,\ G) \\
gen(x,\ \exists yA,\ G) & \textbf{if}\ distinct(x,\ y)\ \&\ gen(x,\ A,\ G) \\
gen(x,\ \forall yA,\ G) & \textbf{if}\ distinct(x,\ y)\ \&\ gen(x,\ A,\ G) \\
gen(x,\ A \vee B,\ G_1 \vee G_2) & \textbf{if}\ gen(x,\ A,\ G_1)\ \&\ gen(x,\ B,\ G_2) \\
gen(x,\ A \wedge B,\ G) & \textbf{if}\ gen(x,\ A,\ G) \\
gen(x,\ A \wedge B,\ G) & \textbf{if}\ gen(x,\ B,\ G) \\
 & \\
con(x,\ P,\ P) & \textbf{if}\ edb(P)\ \&\ free(x,\ P) \\
con(x,\ x=c,\ x=c) & \textbf{if}\ constant(c) \\
con(x,\ A,\ \bot) & \textbf{if}\ \textbf{not}\, free(x,\ A) \\
con(x,\ \neg A,\ G) & \textbf{if}\ pushnot(\neg A,\ B)\ \&\ con(x,\ B,\ G) \\
con(x,\ \exists yA,\ G) & \textbf{if}\ distinct(x,\ y)\ \&\ con(x,\ A,\ G) \\
con(x,\ \forall yA,\ G) & \textbf{if}\ distinct(x,\ y)\ \&\ con(x,\ A,\ G) \\
con(x,\ A \vee B,\ G_1 \vee G_2) & \textbf{if}\ con(x,\ A,\ G_1)\ \&\ con(x,\ B,\ G_2) \\
con(x,\ A \wedge B,\ G) & \textbf{if}\ gen(x,\ A,\ G) \\
con(x,\ A \wedge B,\ G) & \textbf{if}\ gen(x,\ B,\ G) \\
con(x,\ A \wedge B,\ G_1 \vee G_2) & \textbf{if}\ con(x,\ A,\ G_1)\ \&\ con(x,\ B,\ G_2)
\end{array}
$$

Figure 5 Expansion of rules for *gen* and *con* to produce "generators"

□

Lemma 8 2 If F is evaluable, then after Step 1d of Alg 8 1

1 $gen(x,\ \exists{*}G(x) \wedge A(x))$ holds

2 R does not contain x

3 If y is free in $\exists xA$, then $gen(y,\ R)$ holds

Proof It is obvious that $gen(x,\ G(x))$ holds, from which (1) follows

Using the fact that $con(x,\ A)$ holds, it is easy to show by structural induction that during truth value simplification each subformula B of A for which $gen(x,\ B)$ holds evaluates to *false* Thus for all B that do not evaluate to *false*, $con(x,\ B)$ holds and $gen(x,\ B)$ does not That R does not contain x follows easily

Item (3) is easily verified by considering a conservative transformation of A in which the only negations are immediately above atoms By structural induction, it can be shown that for every subformula B such that $gen(y,\ B)$ holds, either B evaluates to *false* or $gen(y,\ B)$ still holds ∎

Lemma 8 3 Let $A(x)$, $G(x)$ and R be as described in Alg 8 1 Then $A(x) \equiv (\exists{*}G(x) \wedge A(x)) \vee R$

Proof Let

$$A_1(x) \stackrel{\text{def}}{=} \exists{*}G(x) \wedge A(x)$$
$$A_2(x) \stackrel{\text{def}}{=} \neg\exists{*}G(x) \wedge A(x)$$

Clearly $A(x) \equiv A_1(x) \vee A_2(x)$ But $R \equiv A_2(x)$ ∎

Theorem 8.4 Every evaluable formula can be effectively transformed into an equivalent allowed formula

Proof By Alg 8 1 and Lemmas 8 2 and 8 3 ∎

It follows immediately from this theorem and Theorem 7 1 that every range restricted formula can also be effectively transformed into an equivalent allowed formula In this special case, our procedure reduces to a slight variant of Decker's, where $\exists{*}G(x)$ plays the role of *range expression* and R is called the *remainder*

Finally, we observe that the expanded rules for *gen* and *con* have some nondeterminacy for conjunctions the G of either conjunct can be adopted when *gen* holds for both This choice represents an opportunity for optimization

9 Translation into a Relational Algebra Expression

We now describe a procedure to translate any allowed formula into an equivalent relational algebra expression. In combination with the transformation of the previous section, this allows any evaluable formula to be translated into an equivalent relational algebra expression.

The translation procedure has two main phases: transformation of the allowed formula into relational algebra normal form, and translation of the normal form into a relational algebra expression.

9.1 Relational Algebra Normal Form

To facilitate defining relation algebra normal form, it is convenient to define two types of formulas.

Definition 9.1 We define D- and G-formulas in terms of atoms and each other as follows:

- A *D-formula* is one of
  - a G-formula
  - $D \wedge \neg G$, where D is a D-formula and G is a G-formula
  - $D \wedge x = y$ or $D \wedge x \neq y$, where D is a D-formula (Recall that $x \neq y$ abbreviates $\neg x = y$.)
  - a conjunction $D_1 \wedge D_2$ of D-formulas
- A *G-formula* is one of
  - an *edb* atom P
  - an atom of the form $x = c$ (treated as an *edb* atom $x \stackrel{q}{=} c$)
  - $\exists y D$, where D is a D-formula containing y
  - a disjunction $G_1 \vee G_2$ of G-formulas

D- and G-subformulas are subformulas that are D- and G-formulas respectively. □

Definition 9.2 A formula F is in *relational algebra normal form* (RANF) if it is a D-formula and

1. For each G-subformula of the form $G_1 \vee G_2$, the same variables are free in G_1 and G_2.
2. For each D-subformula of the form $D \wedge \neg G$, the free variables of G are a subset of the free variables of D.
3. For each D-subformula of the form $D \wedge x = y$ or $D \wedge x \neq y$, x and y are free in D.

□

Lemma 9.1 Every RANF formula is allowed.

Proof Clearly *gen* holds for every free variable in every D- and G-subformula of an RANF formula. ▮

Example 9.1 The converse of Lemma 9.1 is false. Not only are the following allowed formulas not in RANF, but no conservative transformation of them yields an RANF formula.

$$P(x,y) \wedge (Q(x) \vee R(y))$$
$$P(x,y) \wedge \neg\exists z(Q(x,z) \wedge \neg R(y,z))$$
$$P(x) \wedge \neg\exists y(Q(y) \wedge \neg\exists z R(x,y,z))$$

□

9.2 Transformation into RANF

We now present a straightforward algorithm to transform an allowed formula into an equivalent RANF formula. In terms of producing a small RANF equivalent, we acknowledge that this algorithm is not the last word on the subject, but it demonstrates feasibility and is easy to prove correct.

Algorithm 9.1 ranf(F)

INPUT: An allowed formula F

OUTPUT: An RANF formula F_2 equivalent to F

PROCEDURE:

1. Repeatedly apply all possible transformations of the following form:

$$\begin{array}{rcll} \neg\neg A & \longrightarrow & A & \text{(T1)} \\ \neg(A \wedge B) & \longrightarrow & \neg A \vee \neg B & \text{(T2)} \\ \neg(A \vee B) & \longrightarrow & \neg A \wedge \neg B & \text{(T3)} \\ \forall x A(x)) & \longrightarrow & \neg\exists x \neg A(x) & \text{(T4)} \\ \exists x(A(x) \vee B(x)) & \longrightarrow & \exists u A(u) \vee \exists v B(v) & \text{(T9)} \\ A \wedge (B \vee C) & \longrightarrow & (A \wedge B) \vee (A \wedge C) & \text{(T11)} \end{array}$$

   Call the resulting formula F_1.

2. Starting with F_1, repeatedly apply the following transformations from the top down wherever possible.

   For each subformula

$$G \stackrel{\text{def}}{=} C_1 \wedge \dots \wedge C_j \wedge \dots \wedge C_n$$

   where some variable x is free in C_j and $gen(x, C_j)$ does not hold, find a conjunct $C_i(x)$ for which $gen(x, C_i)$ does hold (possible because the formula is allowed). If $i > j$, move C_j just to the right of C_i, but we continue to call the conjunct for which *gen* fails C_j. Now if $C_j \stackrel{\text{def}}{=} \neg\exists y A(x,y)$, then rewrite

$$C_j \stackrel{\text{def}}{=} \neg\exists y A(x,y) \longrightarrow \neg\exists y(C_i(x) \wedge A(x,y))$$

   If G has no free variables, then every conjunct C_i may be negative. In this case, to ensure a D-formula, rewrite

$$G \longrightarrow true \wedge G$$

   Call the resulting formula F_2, and output it.

□

Lemma 9.2 After Step 1 of Alg 9.1, the resulting formula F_1 has the following properties

1. $F_1 \equiv F$ and is allowed
2. F_1 has the form $D_1 \vee \cdots \vee D_m$, where $m \geq 1$ and every D_k has the form described in (3). This is the only place where disjunction occurs in F_1
3. Each D_k in (2) and (4) has the form $C_1 \wedge \cdots \wedge C_n$ ($n \geq 1$ and varies with k), where each C_j has the form of (4)
4. Every C_j in (3) has the form E_j or $\neg E_j$, where E_j is either an atom, or is of the form $\exists y D_k$, where D_k has the form of (3)

Proof Each rewrite rule (Ti) is justified for property (1) by equivalence (Ei) and Theorem 6.6. Since no (Ti) is applicable in F_1, properties (2–4) follow ∎

Lemma 9.3 After Step 2 of Alg 9.1, the resulting formula $F_2 \equiv F_1$, preserves properties (1–4) of Lemma 9.2, and has the following additional property

5. For every subformula $C_1 \wedge \cdots \wedge C_n$ of F that is maximal (i.e., not immediately under another $\wedge$), if x is free in C_j and $gen(x, C_j)$ does not hold, then there exists C_i with $i < j$, for which $gen(x, C_i)$ does hold

Proof The rewrite rule in Step 2 of Alg 9.1 produces an equivalent formula because of the identity $A \wedge \neg B \equiv A \wedge \neg(A \wedge B)$. Property (5) is achieved because the formula being operated upon is always allowed ∎

Theorem 9.4 Alg 9.1 transforms any allowed formula into an equivalent RANF formula

Proof Straightforward from properties (1–5) established in Lemmas 9.2 and 9.3. In particular, if $C_1 \wedge \cdots \wedge C_n$ is a subformula of F, then each prefix $C_1 \wedge \cdots \wedge C_i$ for $i \leq n$ is a D-formula ∎

9.3 From RANF to Relational Algebra

The translation of a formula F in relational algebra normal form into an equivalent relational algebra expression is quite straightforward, the basics are given in [Ull80]. However, it is unnecessary to form the *Dom* relation mentioned there, which includes all constants in query and the database. Because $A \vee B$ only occurs when A and B have the same free variables, we can simply use *union* (possibly after a column permutation). Also, negation only appears as $A \wedge \neg B$, where B's free variables are a subset of A's, permitting the use of a generalized set difference operator

Definition 9.3. The relational operation *generalized set difference*, P **diff** Q, yields the set of tuples in P whose projections are not in Q. That is,

$$P \ \mathbf{diff}\ Q \equiv P - \pi(P \bowtie Q)$$

where the (equi-)join is on the components of Q (which must be a subset of those of P), and the projection is onto the components of P. If P and Q have the same arity, then P **diff** Q is simply $P - Q$, possibly after a permutation of columns □

Although we have defined P **diff** Q in terms of primitive relational operators, it should be implemented as a primitive in its own right, using techniques similar to those used for efficient joins. (In fact we believe that **diff** is also called *anti-join*.) Thus we keep **diff** in our final relational algebra expressions

We assume that the system builds (in effect) a temporary $\stackrel{q}{=}$ relation for constants that appear in the query, and treats $x = c$ as an *edb* predicate $x \stackrel{q}{=} c$

Example 9.2 We show below, for several allowed formulas (*cf* Example 9.1), the RANF and relational algebra expression constructed by the above procedures

$$\begin{array}{ll} & P(x,y) \wedge (Q(x) \vee R(y)) \\ \equiv & (P(x,y) \wedge Q(x)) \vee (P(x,y) \wedge R(y)) \\ \longrightarrow & \pi_{12}(P \bowtie_{1=1} Q) \cup \pi_{12}(P \bowtie_{2=1} R) \end{array}$$

$$\begin{array}{ll} & P(x) \wedge \forall y(\neg Q(y) \vee \exists z R(x,y,z)) \\ \equiv & P(x) \wedge \neg \exists y(P(x) \wedge Q(y) \wedge \neg \exists z R(x,y,z)) \\ \longrightarrow & P - \pi_1(P \times Q - \pi_{12} R) \end{array}$$

$$\begin{array}{ll} & P(x,y) \wedge \forall z(\neg Q(x,z) \vee R(y,z)) \\ \equiv & P(x,y) \wedge \neg \exists z(P(x,y) \wedge Q(x,z) \wedge \neg R(y,z)) \\ \longrightarrow & P - \pi_{12}(\pi_{124}(P \bowtie_{1=1} Q)\ \mathbf{diff}_{2,3=2,3}\ R) \end{array}$$

□

Theorem 9.5 Every allowed formula can effectively be translated into an equivalent relational algebra expression

Proof. Theorem 9.4 and above discussion ∎

Many simplifications of the relational algebra expressions produced by the procedures of this section can be made during their construction. Alternatively, final expressions can be simplified using, e.g., the methods in [Ull80]

10 Relation between Evaluable and Domain Independent Classes

In this section we show that the evaluable class is contained in the domain independent class and that with the restriction to formulas with no repeated predicates *evaluable* is equivalent to *domain independent* To do so, we use the fact that *domain independent* is equivalent to *definite*, which we now define [ND82]

Definition 10 1 Let **I** be an interpretation with domain **D** for a formula F, and let p_i be the relations assigned by **I** to the *edb* predicates P_i that occur in F Let $*$ be a value not in **D** Then the ***-extension** of **I** is the interpretation $\mathbf{I}'$ with domain $\mathbf{D}' = \mathbf{D} \cup \{*\}$ that assigns the same relations p_i to the predicates P_i as does **I** We denote appropriate cross products of **D** and $\mathbf{D}'$ by $\vec{\mathbf{D}}$ and $\vec{\mathbf{D}}'$, respectively □

Definition 10 2: A formula F is called *definite* if, for all interpretations **I**, F is satisfied at the same points in **I** as in $\mathbf{I}'$, where $\mathbf{I}'$ is the $*$-extension of **I** In other words, $\vec{a}$ satisfies F in $\mathbf{I}'$ if and only if $\vec{a}$ satisfies F in **I** □

10 1 Evaluable Formulas are Domain Independent

We now show that every evaluable formula is domain independent This was proved originally in [Dem82] for evaluable formulas as defined there The statement needs to be re-examined because we have used an independent definition, and have incorporated equality

Our proof is significantly simpler because of Theorems 8 4 and 9 4, which state that every evaluable formula has an equivalent RANF formula Hence it is sufficient to prove domain independence for RANF formulas

Lemma 10 1 Let $F(x)$ be a formula, possibly containing other free variables besides x Let **I** be an interpretation for F with domain **D** and $*$-extension $\mathbf{I}'$ If $gen(x, F)$ holds, then F does not hold in $\mathbf{I}'$ for any assignment that assigns $*$ to x

Proof Use induction on formula size, which we define to be the number of atoms plus the number of quantifiers (negations are excluded) For the basis F is an atom and not of the form $x = y$, the conclusion is immediate For the induction, one of the following cases applies

- $F \overset{\text{def}}{=} A \wedge B$ One of A and B satisfies *gen*, and therefore by the inductive hypothesis, does not hold if x is assigned $*$
- $F \overset{\text{def}}{=} A \vee B$ Both of A and B satisfy *gen*, and therefore by the inductive hypothesis, do not hold if x is assigned $*$
- $F \overset{\text{def}}{=} \%yA$ A satisfies *gen*, and therefore by the inductive hypothesis, does not hold if x is assigned $*$
- $F \overset{\text{def}}{=} \neg A$ If A is an atom, the conclusion holds vacuously, since $gen(x, F)$ is false Otherwise, push the $\neg$ down giving G (i e , $pushnot(\neg A, G)$ holds) Now either G is an atom other than $x = y$, or one of the above cases applies to G

∎

Lemma 10 2 If F is an RANF formula, then F is definite

Proof. In view of Lemma 10 1, it is sufficient to show that *gen* holds for all free variables in every D-subformula and in every G-subformula of F This is straightforward by structural induction For example, suppose D is a D-formula If D is of the form $A \wedge \neg B$, then the free variables of B are a subset of those of A, and A is a D-formula Also, if D is of the form $A \wedge x = y$ or $A \wedge x \neq y$, then A is a D-formula in which x and y are free In both cases all the free variables of D are also free in A, and by the inductive hypothesis *gen* holds for them in A, hence in D Other cases are similar ∎

Theorem 10 3 If F is evaluable, then F is definite, and hence is domain independent

Proof. By Theorems 8 4 and 9 4 and Lemma 10 2 ∎

10.2 Evaluable Formulas with No Repeated Predicates

Essentially, the domain independent class is not recursive because a given formula may have a subformula that is superficially not domain independent, but is unsatisfiable, hence (vacuously) domain independent But even though unsatisfiability is decidable for formulas with sufficiently simple quantifier structure [Ack68], we do not consider it practical to test subformulas for unsatisfiability as part of the procedure that transforms them into relational algebra However, formulas in which no predicate symbol is repeated cannot possibly have unsatisfiable subformulas We show that formulas in this class (without equality) are evaluable if and only if they are domain independent This means that any extension to the class of evaluable formulas that remains domain independent must *at least* provide for simplifications based on common subexpressions (e g , subsumption tests), and should probably include some form of inference capability (e g , resolution)

Lemma 10.4· Let F be a formula in prenex-literal normal form (PLNF, see Def 4 1) Let F have no repeated predicate symbols, no equality, and no disjunction If F is not evaluable, then F is not definite The same holds if F has no conjunction

Proof (Sketch) Let $F \stackrel{\text{def}}{=} \%\vec{x}M(\vec{x},\vec{y})$, where

$$M \stackrel{\text{def}}{=} P_1 \wedge \quad \wedge P_n \wedge \neg Q_1 \wedge \quad \wedge \neg Q_m$$

and each P_i and Q_j is an atom of a different predicate Let $\mathbf{D} = \{a\}$ We shall find an interpretation $\mathbf{I}$ with domain $\mathbf{D}$ and $*$-extension $\mathbf{I}'$ such that F evaluates differently in $\mathbf{I}$ and $\mathbf{I}'$ ▮

Theorem 10 5 Let F be a formula with no repeated predicate symbols and no equality Then F is definite if and only if F is evaluable

Proof (Sketch) The "$\Leftarrow$" part holds by Theorem 10 3 above By Cor 6 3 we may assume F is in PLNF, and is given by

$$F \stackrel{\text{def}}{=} \%\vec{x}M(\vec{x},\vec{y})$$

where M is quantifier free We define the size of a formula to be the number of atoms plus the number of quantifiers in it For the "$\Rightarrow$" part, we show by induction on size that if F is definite, then we can reduce to the case covered in Lemma 10 4 ▮

We conjecture that this theorem can be extended to allow some presence of equality However, it cannot be extended much in other directions in view of the fact that (*cf* Example 6 2)

$$F(x) \stackrel{\text{def}}{=} \forall y[(P(x) \wedge Q(y)) \vee (P(x) \wedge \neg R(y))]$$

is domain independent but not evaluable

11 Acknowledgements

We would like to thank Robert Demolombe, who originated the evaluable class of formulas, for helpful discussions and comments on an early draft of this work We also thank Hendrik Decker for helpful discussions

References

[Ack68] W Ackermann *Solvable Cases of the Decision Problem* North-Holland, Amsterdam, 1968

[Dec86] H Decker Integrity enforcement in deductive databases In *1st Int'l Conference on Expert Database Systems*, pages 271–285, 1986

[Dem82] R Demolombe *Syntactical Characterization of a Subset of Domain Independent Formulas* Technical Report, ONERA-CERT, 1982

[DiP69] R A DiPaola The recursive unsolvability of the decision problem for the class of definite formulas *JACM*, 16(2) 324–324, 1969

[Fag80] R Fagin Horn clauses and database dependencies In *12th Ann ACM Symp on Theory of Computing*, pages 123 134, 1980

[Kuh67] J L Kuhns *Answering Questions by Computer A Logical Study* Technical Report RM -5428–PR, Rand Corp , 1967

[LT84] J W Lloyd and R W Topor Making Prolog more expressive *Journal of Logic Programming*, 1(3) 225–240, 1984

[Mak81] J A Makowsky Characterizing data base dependencies In *8th Coll on Automata, Languages and Programming*, Springer Verlag, 1981

[Man74] Z Manna *Mathematical Theory of Computation* McGraw-Hill, New York, 1974

[MUVG86] K Morris, J D Ullman, and A Van Gelder Design overview of the Nail! system In *Third Int'l Conf on Logic Programming*, July 1986

[ND82] J -M Nicolas and R Demolombe *On the Stability of Relational Queries* Technical Report, ONERA-CERT, 1982

[Nic82] J -M Nicolas Logic for improving integrity checking in relational databases *Acta Informatica*, 18(3) 227–253, 1982

[Top86] R Topor *Domain Independent Formulas and Databases* Technical Report 86/11, Univ of Melbourne, 1986 (To appear in *Theoretical Computer Science*)

[Ull80] J D Ullman *Principles of Database Systems* Computer Science Press, Rockville, MD, 1980 (Revised Ed 1982)

A Equality Reduction and Wide Sense Evaluability

In this appendix, we describe transformations that normalize formulas with respect to equality (=), which we call *equality reduction* Many formulas containing equality do not satisfy the requirements for evaluability initially, but are evaluable after equality reduction We say that such formulas are ***evaluable in the wide sense*** Wide sense evaluability is invariant under conservative transformations Since every wide sense evaluable formula is equivalent to an evaluable formula, it is also domain independent

Lemma A 1 Let $F \stackrel{\text{def}}{=} x = t \wedge A(x,t,\vec{y}))$, where t is either a variable or a constant, and is not required to appear in $A(x,t,\vec{y})$ Then

$$F \equiv F' \stackrel{\text{def}}{=} x = t \wedge A(t,t,\vec{y}))$$

$$
\begin{aligned}
F &\stackrel{\text{def}}{=} \exists z[P(x,z) \wedge (x = y \vee Q(x,y,z)) \wedge \neg(z = y \vee R(y,z))] \\
&\equiv \exists z[(z = y \wedge \mathit{false}) \vee (z \neq y \wedge P(x,z) \wedge (x = y \vee Q(x,y,z)) \wedge \neg R(y,z))] \\
&\equiv \exists z[z \neq y \wedge P(x,z) \wedge (x = y \vee Q(x,y,z)) \wedge \neg R(y,z)] \\
&\equiv (x = y \wedge \exists z[z \neq y \wedge P(y,z) \wedge \neg R(y,z)]) \vee (x \neq y \wedge \exists w[w \neq y \wedge P(x,w) \wedge Q(x,y,w) \wedge \neg R(y,w)]) \\
&\equiv (x = y \wedge A(x) \wedge A(y)) \vee (x \neq y \wedge \exists w[w \neq y \wedge P(x,w) \wedge Q(x,y,w) \wedge \neg R(y,w)]) \\
&\quad \textbf{where } A(y) \stackrel{\text{def}}{=} \exists z[z \neq y \wedge P(y,z) \wedge \neg R(y,z)]
\end{aligned}
$$

Figure 6 Equality reduction of a wide sense evaluable formula

Proof In any evaluation, either x is assigned the same value as t or both F and F' evaluate to *false* ∎

The lemma generalizes the transformations (E13–14) in Fig 4 to free variables

Algorithm A 1. Equality Reduction

INPUT A relational calculus formula F

OUTPUT An equivalent equality-reduced formula

PROCEDURE

1 Apply the following transformation wherever possible

Let $A(x)$ be the maximal subformula of F in which x is free A may have other free variables If A contains an atom $x = t$, where t is either another free variable of A or a constant,[5] then

(a) Define $A_1(t)$ to be the formula that results from replacing every occurrence of x in A by t, and then replacing $t = t$ by *true* and carrying out truth value simplification (Def 8 2)

(b) Define $A_2(x)$ to be the formula that results from replacing each occurrence of $x = t$ in $A(x)$ by *false*, and carrying out truth value simplification (Bound variables of A are given different names in A_1 and A_2)

(c) Replace A by

$$A' \stackrel{\text{def}}{=} (x = t \wedge A_1(t)) \vee (x \neq t \wedge A_2(x))$$

(d) If x is bound in F, then replace $\exists x A'$ by

$$A_1(t) \vee \exists x(x \neq t \wedge A_2(x))$$

2 Equality reduction can also be carried out on equalities between two constants, which may be introduced in Step 1 Suppose $c = d$ occurs, where c and d are distinct constants If the system assumes that the distinct name axiom $c \neq d$ is implicit in F, then we can make it explicit at the top level

$$F \longrightarrow c \neq d \wedge F$$

Now replace $c = d$ by *false* throughout F and simplify, as in Step 1b Repeat until all equalities between constants are removed

3 At this point all equalities between two free variables of F that remain can be put in the form of "case splits" at the top of the formula by appropriately "pushing ands" (E11) For any case of the form $x = z \wedge A(z)$, where x is not free in A and $gen(z, A)$ holds, rewrite this case as

$$x = z \wedge A(x) \wedge A(z)$$

This typically arises when A originally contained x but it was substituted for in Step 1 above In an implementation, we would not actually do it this way, we would add a column replication primitive to our relational algebra

□

The correctness of the algorithm follows from Lemma A 1 and elementary arguments

Definition A.1 A formula F is said to be *wide sense evaluable* if Alg A 1 transforms it into an evaluable formula as defined in Def 5 2 □

Example A.1: The formula in Fig 6 is unmotivated, but serves to illustrate the mechanics of the algorithm □

A better characterization of wide sense evaluable formulas is a topic for future research

[5] If A contains $t = x$ such that t equality, transpose it to $x = t$