On minimizing the $\forall$-$\neg$ degree of a formula

Jan Van den Bussche*

August 26, 1997

Abstract

We address the following optimization problem: given a member of a restricted class of predicate logic formulas, give all equivalent formulas in which the number of occurring negations and universal quantifiers is minimal. This problem has applications in the field of nested relational databases. We present algorithms that solve the problem and its associated decision, search and enumeration problems.

1 Introduction

In the field of databases, most research efforts were spent in the context of the relational model of databases [Ull90]. One of the main reasons for the success of relational database systems is that they naturally lend themselves to high-level, declarative query languages.

A fundamentally important query operation is the *selection*, which derives from a relation those tuples that satisfy a certain condition. Selection conditions are typically built from comparisons of the form $A = B$, where A and B are attribute names, and negation and logical connectives.

Recently, next generation database applications became aware of the need of more directly representing complex structures than is possible in the relational model [ABD+89]. In the quest for extending the abilities of relational database systems without losing the advantages of such systems, the

* Address: University of Antwerp (UIA), Dept. Math. & Comp. Science, Universiteitsplein 1, B-2610 Antwerp, Belgium. E-mail: vdbuss@ccu.uia.ac.be.

nested relational model [SS86, TF86] was a particularly elegant step forward. Roughly, the components of the tuples in a nested relation need not be unstructured values, but can be (nested) relations in turn.

Relational query languages can be extended to work on nested relational databases. But, due to the richer data structures provided by nested relations, much more complicated query mechanisms arise naturally. For instance, the selection condition of a selection operation for nested relations can now involve quantifiers. For example, if X and Y are set-valued attributes, $(\forall A \in X)(\forall B \in Y)\neg A = B$ expresses the condition that $X \cap Y = \emptyset$, and $(\exists A \in X)(\forall B \in Y)\neg A = B$ expresses $X \not\subseteq Y$.

Consequently, the implementation strategies for evaluating selection operations, as were developed for traditional relational systems in the seventies [A$^+$76, GAC$^+$79], must be thoroughly reinvestigated and adapted to the nested relational case. In [VdB91], we studied the evaluation of a class of nested relational selections whose condition has the form of a sequence of negations and quantifications, followed by a simple comparison (for example, the conditions given in the previous paragraph). It is shown there that such selection operations can be translated into sequences of elementary ones, which are well optimizable. A similar situation occurs in [OW89], where a translation is described of relational calculus expressions in a "set-oriented" calculus, by which negation and universal quantification are eliminated and expressed instead using set comparisons, which was claimed in [OW89] to have several advantages. Important is that in both cited cases, the complexity of the result of the translation is proportional to the number of occurring negations and universal quantifiers in the input.

Hence, for optimization purposes, it is desirable to preprocess the expressions into an equivalent form, such that this number of occurring negations and universal quantifiers is minimized. In this paper, we address and elaborate further upon this problem. We rephrase the problem independently from its connection with databases. Thus, we are as general as possible, allowing possible other applications too. Moreover, the results presented here may be appreciated in their own right.

Concretely, define a *connective-free* predicate logic formula as a formula having the format of a sequence of negations, existential and universal quantifiers, followed by a fixed predicate. Such formulas naturally correspond to strings over the alphabet $\{\neg, \exists, \forall\}$, and two formulas are *equivalent* if they can be rewritten into each other using the well-known equations $\forall = \neg\exists\neg$,

$\exists = \neg\forall\neg$ and $\neg\neg = \lambda$ (the empty string). Furthermore, the $\forall$-$\neg$ *degree* of a formula is defined as the total number of occurring $\neg$ and $\forall$ in it. For example, the formula

$$(\forall x)(\forall y)\neg p(x, y)$$

is equivalent to

$$\neg(\exists x)(\exists y)p(x, y),$$

which can be alternatively seen by the string rewritings

$$\forall\forall\neg \equiv \forall\neg\neg\forall\neg \equiv \forall\neg\exists \equiv \neg\neg\forall\neg\exists \equiv \neg\exists\exists.$$

The first formula has degree 3, while the second, equivalent one, is more efficient, having only degree 1.

The contents of this paper can be summarized as follows. Preliminaries are given in Section 2. In Section 3, we solve the above mentioned problem, by presenting and proving correct an algorithm that outputs all strings with minimal number of occurring $\neg$ and $\forall$ that are equivalent to a given string. In Section 4 we then observe that in general, the number of such outputs cannot be polynomially bounded. However, it is shown that the associated decision, search and enumeration problems can be solved in low polynomial time.

2 Preliminaries

The following notations will be used throughout. A is the 3-letter alphabet $\{\neg, \exists, \forall\}$ and A^* denotes the monoid of finite strings over A. When talking about strings, we will always mean strings in A^*, unless explicitly stated otherwise. The empty string, over any alphabet, is denoted by λ. Letters i, j, k, ℓ, m, and n stand for natural numbers. For a rational number z, $\lceil z \rceil$ denotes the "ceiling" function applied to z, i.e., the smallest integer m such that $m \geq z$. For any string s, $(s)^k$ denotes s repeated k times, and $|s|$ denotes its length. For a set or list V, $\#V$ denotes the number of elements of V.

We next recall the basic notion of *rewrite system*:

Definition A rewrite system R over A is a set of binary relations over A^*, called *rewrite rules*. Let $s, s' \in A^*$. We denote by $s \mapsto_R s'$ the fact that

$\langle s, s' \rangle$ is in ρ for some rewrite rule ρ of R. So formally, $\mapsto_R = \bigcup_{\rho \in R} \rho$. We say that s is *in normal form* w.r.t. R if there is no s' such that $s \mapsto_R s'$. The transitive closure of $\mapsto_R$ is denoted by $\mapsto_R^+$, and the reflexive and transitive closure by $\mapsto_R^*$. When R is understood from the context, the subscript R will sometimes be omitted. If $s \mapsto_R^* s'$ and s' is in normal form w.r.t. R, then we will say that s' is a *normal form of* s w.r.t. R.

We defined a rewrite rule as a binary relation, which can alternatively be seen as a multivalued or nondeterministic function. Of course, any rewrite system is equivalent to one having only one single rewrite rule, being the union of all rewrite rules in the original system. It will however be convenient to distinguish between several rules in the way as defined above. However, if it does turn out to be the case that a certain rewrite system R indeed consists of a single rewrite rule ρ, then we will, for simplicity, not make the formal distinction between R and ρ.

Let us introduce some rewrite rules that will be important in the sequel:

- $\{\langle \alpha \forall \beta \, , \, \alpha \neg \exists \neg \beta \rangle \mid \alpha, \beta \in A^*\}$ will be denoted by $\forall \to \neg \exists \neg$;
- The inverse of $\forall \to \neg \exists \neg$ will be denoted by $\neg \exists \neg \to \forall$;
- The union of $\forall \to \neg \exists \neg$ and $\neg \exists \neg \to \forall$ will be denoted by $\forall = \neg \exists \neg$;
- $\{\langle \alpha \neg \neg \beta \, , \, \alpha \beta \rangle \mid \alpha, \beta \in A^*\}$ will be denoted by $\neg \neg \to \lambda$;
- The inverse of $\neg \neg \to \lambda$ will be denoted by $\lambda \to \neg \neg$; and
- The union of the last two will be denoted by $\neg \neg = \lambda$.

We will call the rewrite system $\{\forall = \neg \exists \neg, \neg \neg = \lambda\}$ the *standard* rewrite system, and denote it by R_{stand}. If $s \mapsto_{R_{\text{stand}}}^* s'$ then we say that s and s' are *equivalent* and denote this by $s \equiv s'$. For example,

$$\exists \mapsto_{R_{\text{stand}}}^* \neg \neg \exists \neg \neg \mapsto \neg \forall \neg,$$

and hence $\exists \equiv \neg \forall \neg$. Since the rewrite rules of R_{stand} are symmetric, $\equiv$ is indeed an equivalence relation.

We will also use the rewrite system $R_\forall$, defined as $\{\forall \to \neg \exists \neg, \neg \neg \to \lambda\}$. Note that if $s \mapsto_{R_\forall} s'$ then $s \mapsto_{R_{\text{stand}}} s'$. This rewrite system is important for the following reason:

Fact Every string has a unique normal form w.r.t. $R_\forall$.

This observation can be verified by a straightforward induction on the length of the string, and will also follow from Lemma 2, for which we will give a detailed proof.

For $s \in A^*$, the normal form of s w.r.t. $R_\forall$ will be denoted by $s^{R_\forall}$. This normal form can be constructed by performing a left-to-right scan, in which each occurring $\forall$ is replaced by $\neg\exists\neg$, and where pairs of consecutive $\neg$ are eliminated. For example, for $s = \forall\forall\exists$:

$$s \mapsto \neg\exists\neg\forall\exists \mapsto \neg\exists\neg\neg\exists\neg\exists \mapsto \neg\exists\exists\neg\exists = s^{R_\forall}.$$

This leads us to an efficient procedure to decide whether two strings are equivalent, in view of the following, readily verified property:

Fact Let $s, s' \in A^*$. Then $s \equiv s'$ iff $s^{R_\forall} = s'^{R_\forall}$.

As an immediate corollary, it is decidable in linear time whether two given strings are equivalent.

Our topic of main interest is now the following:

Definition Let $s \in A^*$. The $\forall$-$\neg$ *degree* (or *degree* for short) of s, denoted $d(s)$, is the total number of occurring $\forall$ and $\neg$ in s.

For example, $d(\neg\exists\exists\forall\neg) = 3$, but the degree of $\forall\forall\exists$, which is equivalent, is only 2.

The problem that will be addressed in this paper is: *Given $s \in A^*$, find among all equivalent strings those of minimal degree.*

Formally, we define:

Definition Let $s \in A^*$. Then:

$$\mathrm{Min}(s) := \{s' \in A^* \mid s \equiv s' \text{ and } d(s') = \min_{s'' \equiv s} d(s'')\}.$$

3 Finding the equivalent strings of minimal degree

Before specifying our main algorithm, we introduce three more rewrite rules that will be used in this algorithm.

- ρ_1 is the rule consisting of all pairs

$$\langle \alpha\neg(\exists\neg)^k\beta\,,\ \alpha\forall(\exists\forall)^{\frac{k-1}{2}}\beta\rangle,$$

  where k is odd, $\alpha \in A^* - A^*\neg\exists$, and $\beta \in A^* - \exists\neg A^*$.[1]

- ρ_2 is the rule consisting of all pairs

$$\langle \alpha\neg(\exists\neg)^k\beta\,,\ \alpha(\forall\exists)^{\ell_1}\neg(\exists\forall)^{\ell_2}\beta\rangle,$$

  where α, β are as above, k is even, and ℓ_1, ℓ_2 satisfy $2\ell_1 + 2\ell_2 = k$.

- ρ_3 equals $\{\langle \alpha\neg\exists\exists\neg\beta\,,\ \alpha\forall\forall\beta \mid \alpha, \beta \in A^*\}$, and will sometimes also be denoted by $\neg\exists\exists\neg \to \forall\forall$, and its inverse by $\forall\forall \to \neg\exists\exists\neg$.

It is important to note that if $s \mapsto s'$ by any of the above three rules, then $s \equiv s'$. Indeed: ρ_1 and ρ_2 merely apply the rule $\neg\exists\neg \to \forall$ in particular orders, and ρ_3 uses the equivalence

$$\neg\exists\exists\neg \equiv \neg\exists\neg\neg\exists\neg \equiv \forall\forall.$$

Also, it is readily verified that the following holds:

Fact Every string has a unique normal form w.r.t. ρ_1.

For $s \in A^*$, the normal form of s w.r.t. ρ_1 will be denoted by s^{ρ_1}. This normal form can be obtained by rewriting every maximal occurrence of a substring of the form $\neg(\exists\neg)^k$, such that k is odd, into $\forall(\exists\forall)^{\frac{k-1}{2}}$.

Neither ρ_2 nor ρ_3 have the unique normal form property. For ρ_2, every maximal occurrence of a substring of the form $\neg(\exists\neg)^k$, such that k is even, can be rewritten in several ways, the number of which equals the number of different pairs (ℓ_1, ℓ_2) satisfying $2\ell_1 + 2\ell_2 = k$. We will come back to this in Lemma 7. Concerning ρ_3, strings having "overlapping" occurrences of $\neg\exists\exists\neg$ have several normal forms w.r.t. ρ_3, the most simple example being $s = \neg\exists\exists\neg\exists\exists\neg$, which has normal forms $\forall\forall\exists\exists\neg$ and $\neg\exists\exists\forall\forall$.

We are now ready to specify our main algorithm:

Algorithm 1

[1] Informally, α does not end on $\neg\exists$ and β does not start with $\exists\neg$.

Input: $s \in A^*$.

Output: $\mathrm{Min}(s)$.

Method:

1. *Compute* $s_1 := (s^{R_\forall})^{\rho_1}$;
2. *Output all normal forms of* s_1 *w.r.t.* ρ_2;
3. *For each string* s' *output in the previous step, output also all strings* s'' *for which* $s' \mapsto^+_{\rho_3} s''$.

Step 3 can of course be performed simultaneously with step 2. The algorithm can thus alternatively be read as: *output for each normal form* s' *of* $(s^{R_\forall})^{\rho_1}$ *w.r.t.* ρ_2, *all strings* s'' *for which* $s' \mapsto^*_{\rho_3} s''$.

For example, let

$$s = \neg\exists\neg\exists\exists\neg\exists\neg\exists\neg\exists\neg\exists.$$

Then s equals $s^{R_\forall}$. Computing s_1 goes as follows:

$$s \mapsto \forall\exists\exists\neg\exists\neg\exists\neg\exists\neg\exists \mapsto \forall\exists\exists\forall\exists\forall\exists = s^{\rho_1}.$$

s_1 is already in normal form, both w.r.t. ρ_2 and ρ_3, and hence is the only output of the algorithm. We have $d(s_1) = 3$, while $d(s) = 6$.

As another example, consider

$$s = \neg\exists\exists\neg\exists\forall\neg\forall\forall.$$

Then

$$s^{R_\forall} = \neg\exists\exists\neg\exists\neg\exists\neg\exists\exists\neg,$$

which is already in normal form w.r.t. ρ_1. One possible output of step 2 then is

$$\neg\exists\exists\neg\exists\forall\exists\exists\neg,$$

by a single application of ρ_2, with $k = 2$, $\ell_1 = 0$ and $\ell_2 = 1$. In step 3 this output gives rise to another output,

$$\forall\forall\exists\forall\exists\exists\neg.$$

Both outputs have degree 4, while the degree of the original s is 6.

The remainder of this section is devoted to the proof of:

Theorem Algorithm 1 is correct.

Thereto, we first need to introduce a number of auxiliary notions.

Definition Let $s \in A^*$. The number of quantifiers occurring in s is denoted by n_s. For $1 \leq i \leq n_s$, $q_i(s)$ denotes the i-th occurrence of a quantifier in s. For $1 \leq i < n_s$, the substring of s starting at $q_i(s)$ and ending at $q_{i+1}(s)$ is denoted by $r_i(s)$. If $n_s \neq 0$, this notation can be naturally extended to include $r_0(s)$ and $r_{n_s}(s)$: the former denotes the prefix of s up to $q_1(s)$, the latter denotes the suffix of s starting from $q_{n_s}(s)$. Finally, if $n_s = 0$ (i.e., if s is a sequence of $\neg$'s) then we simply put $r_0(s) := s$.

We now associate with s a string $b(s)$ over $\{0, 1\}$:

$$b(s) = b_0(s) \dots b_{n_s}(s),$$

with $b_i(s) := d(r_i(s)) \bmod 2$, for $0 \leq i \leq n_s$.

Note that $|b(s)| = n_s + 1$. For example, if $s = \forall\neg\exists\forall\exists$, then $n_s = 4$, $r_1(s) = \forall\neg\exists$, $r_4(s) = \exists$, and $b(s) = 10110$.

Remark also that $s \equiv s'$ implies $n_s = n_{s'}$. The following important property now holds:

Lemma 2 *Let $s, s' \in A^*$. Then $s \equiv s'$ iff $b(s) = b(s')$.*

Proof: The *only-if* is readily verified. To see the *if*, we use induction on n_s $(= n_{s'})$. If $n_s = 0$, then $s = (\neg)^k$ and $s' = (\neg)^{k'}$ for some k, k' such that $k \bmod 2 = k' \bmod 2$. Clearly, $s \mapsto^*_{R_{\text{stand}}} s'$, using the rewrite rule $\neg\neg = \lambda$. If $n_s = N + 1$, then $s = \alpha Q(\neg)^k$ and $s' = \alpha' Q'(\neg)^{k'}$ for some $\alpha, \alpha' \in A^*$ and quantifiers Q, Q', with $n_\alpha = n_{\alpha'} = N$. We distinguish two cases:

1. $Q = Q'$. Then, since $b(s) = b(s')$, we have $b(\alpha) = b(\alpha')$, whence $\alpha \equiv \alpha'$, by the induction hypothesis. Furthermore, $k \bmod 2 = k' \bmod 2$, whence $(\neg)^k \equiv (\neg)^{k'}$, again by the induction hypothesis. So we conclude that $s \equiv s'$.

2. $Q \neq Q'$. By symmetry we can assume $Q = \exists$, $Q' = \forall$. By the rewrite rule $\forall \to \neg\exists\neg$, we have $s' \equiv s'_1 = \alpha'\neg\exists(\neg)^{k'+1}$. Clearly, $b(s'_1) = b(s')$ and hence, by case 1., we have $s \equiv s'_1 \equiv s'$. ■

This property can also serve as basis of an alternative to the decision procedure for equivalence mentioned in Section 2.

From the structure of $b(s)$ we can get information about the behaviour of Algorithm 1 on input s. To see this, we define:

Definition Let $s \in A^*$. Then the list of maximal occurrences of non-empty substrings of $b(s)$ that consist completely of 1's is denoted by $\mathcal{B}(s)$. These occurrences are ordered in $\mathcal{B}(s)$ as they appear in s.

For example, for

$$s = \exists\neg\exists\exists\exists\neg\exists\neg\exists\exists\exists\exists\neg\exists\neg\exists\neg\exists\exists\exists\neg,$$

we have $b(s) = 010011000111001$ and $\mathcal{B}(s) = \langle 1, 11, 111, 1\rangle$. Note that $s = {}_{s^{R_\forall}}$.

Now recall that every application of the rewrite rules ρ_1 and ρ_2 used in the algorithm involves a pair $\langle \alpha\gamma\beta\,,\, \alpha\gamma'\beta\rangle$, where γ is a maximal occurrence of a substring of $s^{R_\forall}$ of the form $\neg(\exists\neg)^k$. Such occurrences precisely correspond to elements a in $\mathcal{B}(s)$. E.g., in the above example, with $a = 1$ corresponds $\gamma = \neg$, and with $a = 111$ corresponds $\gamma = \neg\exists\neg\exists\neg$. More specifically, if a in $\mathcal{B}(s)$ starts at position $b_p(s)$ in $b(s)$, then the corresponding occurrence in $s^{R_\forall}$ is $\neg(\exists\neg)^k$, starting in $r_p(s^{R_\forall})$, with $k = |a| - 1$. This correspondence will be frequently employed in the sequel.

We will also need two more lemmas:

Lemma 3 *Let $s \in A^*$. Then* $\mathrm{Min}(s)$ *contains a string not containing consecutive occurrences of* $\forall$.

Proof: Let $s' \in \mathrm{Min}(s)$ be arbitrarily chosen. If s' has no consecutive occurrences of $\forall$, then we are done. Otherwise, consider a normal form of s' w.r.t. $\forall\forall \to \neg\exists\exists\neg$. This string has no consecutive occurrences of $\forall$, has the same degree as the original s', is also equivalent to s', and hence is in $\mathrm{Min}(s)$. ■

Lemma 4 *Let $t \in A^*$, such that $b(t) = 1^{|b(t)|}$ (i.e., $b(t)$ consists completely of 1's). Then:*

1. $d(t) \geq \lceil |b(t)|/2 \rceil$.
2. *If $d(t) = \lceil |b(t)|/2 \rceil$, then*

$$t = \begin{cases} \forall(\exists\forall)^{\frac{n_t-1}{2}} & \text{if } n_t \text{ is odd;} \\ (\forall\exists)^{\ell_1}\neg(\exists\forall)^{\ell_2} & \text{if } n_t \text{ is even,} \end{cases}$$

where $2\ell_1 + 2\ell_2 = n_t$.

Proof:

1. By induction on $|b(t)| = n_t + 1$. If $|b(t)| = 1$, then $d(t) \geq 1 = \lceil 1/2 \rceil$. If $|b(t)| = N+1$ (with $N \geq 1$), then t can be written either as $\alpha\exists\neg^k$, with $k \geq 1$ and odd, or as $\alpha\forall\neg^k$, with k even. In both cases, $n_\alpha = N - 1$. We have

$$d(t) \geq d(\alpha) + 1 \geq \lceil b(\alpha)/2 \rceil + 1 = \lceil N/2 \rceil + 1 \geq \left\lceil \frac{N+1}{2} \right\rceil = \lceil |b(t)|/2 \rceil.$$

2. For any $i : 0 \leq i \leq n_t$, we have that $d(r_i(t)) \geq 1$ and odd. But then by the previous item, $d(r_i(t))$ must also be minimal with this property, and therefore $d(r_i(t)) = 1$. It is now straightforward to see that t must be of the form stated in the proposition.

■

We are now ready to prove soundness of our algorithm:

Proposition 5 *Algorithm 1 outputs only strings in* $\mathrm{Min}(s)$.

Proof: Since the rewritings by ρ_3 in step 3 of the algorithm are degree-preserving, it suffices to verify the claim for outputs of step 2. To keep notation simple, we can assume that $s = s^{R_\forall}$. We use induction on $\#\mathcal{B}(s)$. If $\#\mathcal{B}(s) = 0$, then $s = (\exists)^{n_s}$, which is also the only output, and is clearly of minimal degree. Now consider an input string s for which $\#\mathcal{B}(s) = N + 1$, and let s' be an output string of step 2 of the algorithm. We have to show that $s' \in \mathrm{Min}(s)$, for which it is sufficient to show that for some $s'' \in \mathrm{Min}(s)$, $d(s'') \geq d(s')$.

First we introduce some notations. Let $\mathcal{B}(s) = \langle a_1, \ldots, a_N, a_{N+1} \rangle$, and let a_{N+1} start at $b_p(s)$ in $b(s)$. Then $s = \alpha\neg(\exists\neg)^k(\exists)^m$, where:

- α is the prefix of s up to $r_{p-1}(s)$, or, if $p = 0$, $\alpha = \lambda$. So $n_\alpha = p$. Moreover, α does not end in $\neg\exists$. So either $\alpha = \lambda$, $\alpha = \exists$, or α has the form $\beta\exists\exists$.
- $k = |a_{N+1}| - 1$;
- m is the number of 0's following a_{N+1} in $b(s)$.

Correspondingly, $s' = \alpha'\gamma'(\exists)^m$, where:

- α' is defined similarly as α, but now for s' instead of for s. Moreover, α' is an output of step 2 of the algorithm on input α. In particular, $n_{\alpha'} = n_\alpha$.
- $\gamma' = \begin{cases} \forall(\exists\forall)^{\frac{k-1}{2}} & \text{if } k \text{ is odd;} \\ (\forall\exists)^{\ell_1}\neg(\exists\forall)^{\ell_2} & \text{if } k \text{ is even,} \end{cases}$ where $2\ell_1 + 2\ell_2 = k$.
  Note that $n_{\gamma'} = k$.

Now let $s'' \in \mathrm{Min}(s)$. Note that $s \equiv s' \equiv s''$, and hence $b(s) = b(s') = b(s'')$. By Lemma 3, we may also assume that s'' contains no consecutive $\forall$'s. But then s'' can be written as $s'' = \alpha''\gamma''(\exists)^m$, where α'' equals λ or $\exists$ or is of the form $\beta''\exists\exists$, depending on the corresponding form of α. In particular, $n_{\alpha''} = n_{\alpha'} = n_\alpha$. Note that $b(\alpha) = b(\alpha') = b(\alpha'')$, and hence $\alpha \equiv \alpha' \equiv \alpha''$.

Since $d(s') = d(\alpha') + d(\gamma')$ and $d(s'') = d(\alpha'') + d(\gamma'')$, if we can show that $d(\alpha'') \geq d(\alpha')$ and $d(\gamma'') \geq d(\gamma')$, we will have obtained that $d(s'') \geq d(s')$.

Since α' is an output of the algorithm on input α, and $\alpha'' \equiv \alpha$, by the induction hypothesis (given that $\#\mathcal{B}(\alpha) = N$) we have $d(\alpha') \leq d(\alpha'')$.

It remains to show that $d(\gamma'') \geq d(\gamma')$. We know that $b(\gamma) = b(\gamma') = b(\gamma'') = a_{N+1}$. By Lemma 4, item 1., we must have that $d(\gamma'') \geq \lceil |a_{N+1}|/2 \rceil$, and since $d(\gamma') = \lceil |a_{N+1}|/2 \rceil$, as is readily verified, the proof is complete. ■

Having shown soundness, showing completeness is now a lighter task:

Proposition 6 *Algorithm 1 outputs all strings in* $\mathrm{Min}(s)$.

Proof: Consider first the following subclaim: *Each* $s' \in \mathrm{Min}(s)$ *which contains no consecutive occurrences of* $\forall$ *is output by the algorithm in step 2.* This claim can be shown along the lines of the proof of Proposition 5, using Lemma 4. We leave the details to the reader. Now consider $s' \in \mathrm{Min}(s)$ having consecutive occurrences of $\forall$. Let s'' be a normal form of s' w.r.t. $\forall\forall \to \neg\exists\exists\neg$. Then $s'' \mapsto^+_{\rho_3} s'$. So, since by the above subcase, s'' is output by the algorithm in step 2, s' is output by the algorithm in step 3. Hence, Algorithm 1 outputs all strings in $\mathrm{Min}(s)$. ■

The correctness of Algorithm 1 now follows immediately from Propositions 5 and 6.

4 Complexity analysis

In this section, we look at the complexity of the problem of finding equivalent strings of minimal degree. We do this by considering the associated decision, search and enumeration problems.

A first observation is that Algorithm 1 cannot in general run in polynomial time, since its number of outputs cannot in general be bounded by a polynomial. To see this, we define:

Definition Let $s \in A^*$. Then the list of maximal occurrences of substrings of $s^{R_\forall}$ of the form $\neg(\exists\neg)^k$, such that k is even, is denoted by $\mathcal{C}(s)$. The occurrences are ordered in $\mathcal{C}(s)$ as they appear in $s^{R_\forall}$.

The following property was already suggested in the previous section:

Lemma 7 *Let $s \in A^*$. Then the number of outputs of step 2 of Algorithm 1 on input s equals*

$$\prod_{t \mathrm{in} \mathcal{C}(s)} (\frac{n_t}{2} + 1).$$

Proof: It is readily seen on inspection of the algorithm that the number of outputs of step 2 equals

$$\prod_{t \mathrm{in} \mathcal{C}(s)} N_2(t),$$

where $N_2(t)$ is the number of different normal forms of t w.r.t. ρ_2. This number equals the number of different pairs (ℓ_1, ℓ_2) for which $2\ell_1 + 2\ell_2 = n_t$: clearly, this is precisely $n_t/2 + 1$ (n_t is even.) ■

If we define $\tilde{\mathcal{C}}(s)$ to be the subset of $\mathcal{C}(s)$, consisting of those elements t for which $n_t \geq 2$, it follows that on input s, Algorithm 1 outputs at least $2^{\#\tilde{\mathcal{C}}(s)}$ strings. In view of this, the only hope to have a polynomial upper bound on the number of outputs of Algorithm 1 is that $\#\tilde{\mathcal{C}}(s)$ is logarithmic in $|s|$. But this fails to be true: there is an infinite class of strings for which $\#\tilde{\mathcal{C}}(s)$ is linear in $|s|$. Indeed, for an arbitrary $m > 0$, define

$$s_m = (\neg\exists\neg\exists\neg\exists\exists)^{m-1}\neg\exists\neg\exists\neg.$$

Then $\#\tilde{\mathcal{C}}(s_m) = m$ and $|s_m| = 7m - 2$.

Concluding, the problem of finding all equivalent strings of minimal degree is of superpolynomial output complexity. However, it is not difficult

to see that Algorithm 1 can be implemented in time linear in its number of outputs. Remark also that for each $s' \in \mathrm{Min}(s)$, $|s'|$ is bounded by $2n_s + 1 \leq 2|s| + 1$, since a necessary condition for s' to be minimal is that no consecutive $\neg$'s occur in s'. This suggests that the problem is not inherently intractable. We will confirm this by next showing that the associated decision, search, and enumeration problems can be solved in low polynomial time.

Devising linear-time algorithms for the search (Algorithm 8) and decision (Algorithm 9) problem is easy:

Algorithm 8 (Search problem)

Input: $s \in A^*$.

Output: *An* $s' \in Min(s)$.

Method: *Output the particular normal form of* $s^{R_\forall}$ *w.r.t.* $\neg\exists\neg \rightarrow \forall$, *obtained by applying the rule in a single scan from left to right.*

Algorithm 9 (Decision problem)

Input: $s, s' \in A^*$.

Output: *Is* $s' \in Min(s)$*?*

Method: *First, test whether* $s \equiv s'$, *using the procedure mentioned in Section 2. If not equivalent, output false. Otherwise, apply the search algorithm, yielding a string* $s'' \in Min(s)$. *Output the boolean* $d(s') = d(s'')$.

We have:

Proposition Algorithms 8 and 9 are correct and run in linear time.

Proof: First consider the search algorithm. During the repeated left-to-right application of $\neg\exists\neg \rightarrow \forall$, maximal occurrences of $\neg(\exists\neg)^k$, such that k is odd, are rewritten into $\forall(\exists\forall)^{\frac{k-1}{2}}$, which corresponds to applying ρ_1, while maximal occurrences of $\neg(\exists\neg)^k$, such that k is even, are rewritten into $(\forall\exists)^k\neg$, which corresponds to applying ρ_2 with $\ell_1 = k/2$ and $\ell_2 = 0$. Hence, the output of the search algorithm is also an output of Algorithm 1, in step 2. Clearly,

only linear time is needed for computing $s^{R_\forall}$ and performing the left-to-right scan.

The correctness and linear time complexity of the decision algorithm now follow immediately. ■

The enumeration problem is more complicated. Given s, our task is to compute $\#\mathrm{Min}(s)$. Of course, a naive way to do this is to run Algorithm 1, while increasing a counter each time a string is output. But obviously, a serious drawback of this enumeration procedure is that it takes superpolynomial time, as we saw above. What we want is an exact figure for $\#\mathrm{Min}(s)$, formally independent of Algorithm 1, and computable in polynomial time.

In Lemma 7, we already found how to compute the cardinality of the subset of $\mathrm{Min}(s)$ consisting of those strings output in step 2 of Algorithm 1. Now we must refine this calculation by taking also step 3 of the algorithm into account. Recall that in step 3, random occurrences of $\neg\exists\exists\neg$ are rewritten into $\forall\forall$. Our analysis will be built around the following auxiliary notion:

Definition Let $s \in A^*$. Then $\mathcal{D}(s)$ is the list of maximal occurrences of substrings of $s^{R_\forall}$ of the form $(t_1\exists\exists)\ldots(t_m\exists\exists)t_{m+1}$, where t_i is in $\mathcal{C}(s)$, for $1 \leq i \leq m$. The *weight* of such a substring is defined to be m. The occurrences are ordered in $\mathcal{D}(s)$ as they appear in $s^{R_\forall}$.

For example, in the following string s, $\mathcal{D}(s) = \langle L_1, L_2 \rangle$:

$$s = s^{R_\forall} = \underbrace{\neg\exists\neg\exists\neg\exists\neg}_{L_1}\exists\exists\exists\underbrace{\neg\exists\neg\exists\neg\exists\exists\neg\exists\exists\neg}_{L_2}\exists\exists\neg\exists\neg.$$

The weight of L_1 is 0; that of L_2 is 2.

The weight of an element of $\mathcal{D}(s)$ equals exactly the number of occurrences of $\neg\exists\exists\neg$ in it. Moreover, the occurrences of $\neg\exists\exists\neg$ in (elements of) $\mathcal{D}(s)$ match exactly those in $(s^{R_\forall})^{\rho_1}$. The reader is invited to check this claim on the above example. This leads to:

Lemma 10 *Let $s \in A^*$. Then*

$$\#\mathrm{Min}(s) = \prod_{L \,\mathrm{in}\, \mathcal{D}(s)} \#\mathrm{Min}(L).$$

A formal proof of this lemma is straightforward and left to the reader. By the lemma, we can further concentrate on individual members of $\mathcal{D}(s)$, or

$s' \in \mathrm{Min}(s)$	$\sigma_s(s')$
$\forall\exists\neg\exists\exists\neg\exists\exists\neg$	00
$\neg\exists\forall\exists\exists\neg\exists\exists\neg$	00
$\forall\exists\forall\forall\exists\exists\neg$	10
$\forall\exists\neg\exists\exists\forall\forall$	01
$\neg\exists\forall\exists\exists\forall\forall$	01

Table 1: $\sigma_s(s')$ for each $s' \in \mathrm{Min}(s)$, where $s = \neg\exists\neg\exists\neg\exists\exists\neg\exists\exists\neg$.

somewhat more general, on strings of the form $(t_1\exists\exists)\dots(t_m\exists\exists)t_{m+1}$, where $t_i = \neg(\exists\neg)^{k_i}$ with k_i even, for $1 \leq i \leq m+1$. Note that for a string s of this type, $(s^{R_\forall})^{\rho_1} = s$, so we can ignore step 1 of Algorithm 1.

With $s = (t_1\exists\exists)\dots(t_m\exists\exists)t_{m+1}$ as above, we can associate a sequence $\mathcal{N}(s)$ of $m+1$ numbers defined by:

$$\mathcal{N}(s) := \langle k_1/2+1\,,\dots,\,k_{m+1}/2+1\rangle$$

(compare with Lemma 7, noting that $k_i = n_{t_i}$). Further, with each $s' \in \mathrm{Min}(s)$, we associate a *boolean string* $\sigma_s(s')$ over the alphabet $\{T(rue), F(alse)\}$ of length m, defined by:

$$\sigma_s(s') = \sigma_s(s')_1 \dots \sigma_s(s')_m,$$

where $\sigma_s(s')_i$ is *True* iff in the computation of s' by Algorithm 1 applied to s, the i-th occurrence of $\neg\exists\exists\neg$ in s is preserved in step 2 and is rewritten into $\forall\forall$ in step 3 of the algorithm.

For example, if $s = \neg\exists\neg\exists\neg\exists\exists\neg\exists\exists\neg$, then $\mathcal{N}(s) = \langle 2,1,1\rangle$, and Table 1 shows $\sigma_s(s')$ for each $s' \in \mathrm{Min}(s)$.

Not all boolean strings equal $\sigma_s(s')$ for some s'. Actually, we can precisely characterize the range of σ_s, as done in the next lemma, the proof of which first requires a definition:

Definition Let z be a boolean string of length m. A position $i : 1 < i < m+1$ is called *free* w.r.t. z if both the $i-1$-th and the i-th symbol of z are *False*. If $m \neq 0$, this notion can be naturally extended to include 1 and $m+1$: 1 (resp. $m+1$) is free w.r.t. z if the first (resp. the last) symbol of z is *False*. Finally, if $m = 0$ (i.e., if z is the empty string), then we simply state that 1 is free w.r.t. z. The set $\{i : 1 \leq i \leq m+1 \mid i \text{ free w.r.t. } z\}$ is denoted by $\mathrm{Free}(z)$.

Lemma 11 *Let s equal $(t_1\exists\exists)\ldots(t_m\exists\exists)t_{m+1}$ as above, and let z be a boolean string of length m. Then $z = \sigma_s(s')$ for some $s' \in \mathrm{Min}(s)$ iff z contains no consecutive occurrences of True.*

Proof: *Only if:* For simplicity, we assume that $m = 2$; the general argument is analogous. We must show that there is no $s' \in \mathrm{Min}(s)$ for which $\sigma_s(s') = TT$. Suppose that there is such a s'. Then in the computation of s' from s by Algorithm 1, in step 2, the first as well as the second occurrence of $\neg\exists\exists\neg$ must be preserved, in order to be rewritten into $\forall\forall$ in step 3. For the first occurrence to be preserved, t_2 must be rewritten into $\neg(\exists\forall)^{k_2/2}$ in step 2; however, for the second to be preserved, t_2 must be rewritten into $(\forall\exists)^{k_2/2}\neg$. If $k_2 > 0$, this yields a contradiction. If $k_2 = 0$, then $s = t_1\exists\exists\neg\exists\exists t_3$, and regardless of how t_1 and t_3 are rewritten in step 2, at most one application of $\neg\exists\exists\neg \rightarrow \forall\forall$ will be possible in step 3, a contradiction with $\sigma_s(s') = TT$.

If: For each $i : 1 \leq i \leq m+1$, define t'_i as follows. If $i \in \mathrm{Free}(z)$, then t'_i may be any normal form of t_i w.r.t. ρ_2. Otherwise, we consider the following possibilities.

- If $i = 1$, then $t'_i := (\forall\exists)^{k_1/2}\neg$.
- If $i = m+1$, then $t'_i := \neg(\exists\forall)^{k_{m+1}/2}$.
- Otherwise, if the $i-1$-th symbol of z is *True*, then $t'_i := \neg(\exists\forall)^{k_i/2}$.
- Otherwise, the i-th symbol of z is *True*, and $t'_i := (\forall\exists)^{k_i/2}\neg$.

All the above possibilities are mutually exclusive. Now define $s'' := (t'_1\exists\exists)\ldots(t'_m\exists\exists)t'_{m+1}$, and define s' as the string obtained from s'' by repeatedly applying $\neg\exists\exists\neg \rightarrow \forall\forall$ according to z. s' is well-defined, since z contains no consecutive occurrences of *True*. Clearly, $s' \in \mathrm{Min}(s)$ and $\sigma_s(s') = z$. ■

We will denote the set of all boolean strings of length m having no consecutive occurrences of *True* by $\mathcal{E}_m$. We point out that $\#\mathcal{E}_m$ equals F_{m+2}, the $m+2$-th Fibonacci number.

In order now to tie all the above notions together, we introduce one more auxiliary construct. Consider an arbitrary $m+1$-ary sequence of numbers $\mathcal{R} = \langle r_i \rangle_{1 \leq i \leq m+1}$, and an arbitrary boolean string z of length m. Then the *product* of $\mathcal{R}$ and z is defined by:

$$\mathcal{R} \times z := \prod_{i \in \mathrm{Free}(z)} r_i.$$

We can now show:

Lemma 12 *Let s be as above, and let $z \in \mathcal{E}_m$. Let $\sigma_s^{-1}(z)$ denote $\{s' \in \mathrm{Min}(s) \mid \sigma_s(s') = z\}$. Then $\#\sigma_s^{-1}(z) = \mathcal{N}(s) \times z$.*

Proof: By induction on m. If $m = 0$, then $s = t_1$, and $\mathrm{Min}(s) = \{(\forall\exists)^{\ell_1}\neg(\exists\forall)^{\ell_2} \mid 2\ell_1 + 2\ell_2 = k_1\}$. Furthermore, z must be the empty string, and $\sigma_s^{-1}(z) = \mathrm{Min}(s)$. So

$$\#\sigma_s^{-1}(z) = \#\mathrm{Min}(s) = k_1/2 + 1 = \mathcal{N}(s) \times z.$$

If $m > 0$, then s can be written as $\alpha\exists\exists t_{m+1}$, where $\alpha = (t_1\exists\exists)\ldots(t_{m-1}\exists\exists)t_m$. We write z as $z_1 \ldots z_m$, where each z_i is *True* or *False*. By the induction hypothesis, $\#\sigma_\alpha^{-1}(z_1 \ldots z_{m-1}) = \mathcal{N}(\alpha) \times z_1 \ldots z_{m-1}$. We now consider two cases.

If $z_m = \mathit{False}$, then for each s' for which $\sigma_s(s') = z$, in the computation of s' from s by Algorithm 1, t_{m+1} can be rewritten into any one of $\{(\forall\exists)^{\ell_1}\neg(\exists\forall)^{\ell_2} \mid 2\ell_1 + 2\ell_2 = k_{m+1}\}$ in step 2. This yields $k_{m+1}/2 + 1$ possibilities, and we have

$$\#\sigma_s^{-1}(z) = (\tfrac{k_{m+1}}{2} + 1)\#\sigma_\alpha^{-1}(z_1 \ldots z_{m-1}) = (\tfrac{k_{m+1}}{2} + 1)\mathcal{N}(\alpha) \times z_1 \ldots z_{m-1} = \mathcal{N}(s) \times z.$$

If $z_m = \mathit{True}$, then for each s' for which $\sigma_s(s') = z$ it holds that in the computation of s' from s by Algorithm 1, t_{m+1} is necessarily rewritten into $\neg(\exists\forall)^{k_{m+1}/2}$ in step 2. It follows that

$$\#\sigma_s^{-1}(z) = \#\sigma_\alpha^{-1}(z_1 \ldots z_{m-1}) = \mathcal{N}(\alpha) \times z_1 \ldots z_{m-1} = \mathcal{N}(s) \times z.$$

■

By the combined efforts of Lemmas 10, 11 and 12, we have thus established:

Proposition 13 *Let $s \in A^*$. Then*

$$\#\mathrm{Min}(s) = \prod_{L \in \mathrm{Lin}\mathcal{D}(s)} \sum_{z \in \mathcal{E}_{w(L)}} \mathcal{N}(L) \times z,$$

where for each L in $\mathcal{D}(s)$, $w(L)$ denotes the weight of L.

Although we now have an exact figure for $\#\mathrm{Min}(s)$ formally independent of Algorithm 1, we still have not arrived at our final goal, being an efficient (polynomial time) algorithm for computing $\#\mathrm{Min}(s)$. Indeed, in the product of Proposition 13, each factor is a sum over $\#\mathcal{E}_{w(L)}$ terms, the naive implementation of which may require an exponential number of steps, since $w(L)$ may be linear in $|s|$ and $\#\mathcal{E}_m = F_{m+2}$.[2]

Fortunately, we have the following property. For an arbitrary sequence of $m+1$ numbers $\mathcal{R} = \langle r_i \rangle_{1 \le i \le m+1}$, denote the sum $\sum_{z \in \mathcal{E}_m} \mathcal{R} \times z$ by $\Sigma(\mathcal{R})$. Then the following Fibonacci-like recurrence holds for $\Sigma(\mathcal{R})$:

Lemma 14

$$\begin{array}{ll} m = 0: & \Sigma(\langle r_1 \rangle) = r_1; \\ m = 1: & \Sigma(\langle r_1, r_2 \rangle) = r_1 r_2 + 1; \\ m \ge 2: & \Sigma(\langle r_1, \ldots, r_{m+1} \rangle) = r_1 \Sigma(\langle r_2, \ldots, r_{m+1} \rangle) + \Sigma(\langle r_3, \ldots, r_{m+1} \rangle). \end{array}$$

Proof: If $m = 0$, then $\mathcal{E}_m = \{\lambda\}$, and $\langle r_1 \rangle \times \lambda = r_1$. If $m = 1$, then $\mathcal{E}_m = \{0, 1\}$, and $\langle r_1, r_2 \rangle \times 0 + \langle r_1, r_2 \rangle \times 1 = r_1 r_2 + 1$. Now let $m \ge 2$. We have

$$\mathcal{E}_m = \mathcal{E}_{m,1} \sqcup \mathcal{E}_{m,2},$$

where $\mathcal{E}_{m,1}$ ($\mathcal{E}_{m,2}$) is the subset of $\mathcal{E}_m$ consisting of those strings whose first symbol is *False* (*True*).[3]

If we write $z = z_1 \ldots z_m$, where each z_i is *True* or *False*, then clearly:

- if $z \in \mathcal{E}_{m,1}$:

$$\mathcal{R} \times z = r_1 \cdot (\langle r_2, \ldots, r_{m+1} \rangle \times z_2 \ldots z_m),$$

  and $z_2 \ldots z_m$ is an arbitrary element of $\mathcal{E}_{m-1}$;

- if $z \in \mathcal{E}_{m,2}$:

$$\mathcal{R} \times z = \langle r_3, \ldots, r_{m+1} \rangle \times z_3 \ldots z_m,$$

  and $z_3 \ldots z_m$ is an arbitrary element of $\mathcal{E}_{m-2}$.

[2] Recall that F_n is asymptotically exponential in n.

[3] Note that $\#\mathcal{E}_{m,1} = \#\mathcal{E}_{m-1}$ and $\#\mathcal{E}_{m,2} = \#\mathcal{E}_{m-2}$, which gives us, as mentioned earlier, that $\#\mathcal{E}_m = F_{m+2}$, since $\#\mathcal{E}_0 = 1 = F_2$ and $\#\mathcal{E}_1 = 2 = F_3$.

Hence,

$$\begin{aligned}\Sigma(\mathcal{R}) &= \sum_{z \in \mathcal{E}_{m,1}} \mathcal{R} \times z + \sum_{z \in \mathcal{E}_{m,2}} \mathcal{R} \times z \\ &= r_1 \Sigma(\langle r_2, \ldots, r_{m+1} \rangle) + \Sigma(\langle r_3, \ldots, r_{m+1} \rangle).\end{aligned}$$

■

Note that using the recurrence of the above lemma, we can compute $\Sigma(\mathcal{R})$ in time proportional to m.

We can finally present:

Algorithm 15 (Enumeration problem)

Input: $s \in A^*$.

Output: $\#\mathrm{Min}(s)$.

Method: *Compute the product shown in Proposition 13, where each factor,* $\Sigma(\mathcal{N}(L))$, *is computed using Lemma 14.*

Proposition Algorithm 15 is correct and runs in quadratic time.

Proof: The correctness was already shown by Proposition 13. The quadratic time complexity follows from the following facts:

1. $\#\mathcal{D}(s) \leq |s|$;
2. for each L in $\mathcal{D}(s)$, $w(L) \leq |s|$;
3. computing $\mathcal{N}(L)$ takes only linear time; and
4. computing $\Sigma(\mathcal{N}(L))$ using Lemma 14 takes only time proportional to $w(L)$.

■

Acknowledgement

Thanks go to Jan Paredaens, for inspiring discussions.

References

[A+76] M.M. Astrahan et al. System R: a relational approach to data management. *ACM Transactions on Database Systems*, 1(2):96–137, 1976.

[ABD+89] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The object-oriented database system manifesto. In W. Kim, J.-M. Nicolas, and S. Nishio, editors, *Proceedings 1st International Conference on Deductive and Object-Oriented Databases*, pages 40–57. Elsevier Science Publishers, 1989.

[GAC+79] P. Griffiths, M. Astrahan, D. Chamberlin, R. Lorie, and T. Price. Acces path selection in a relational database management system. In P. Bernstein, editor, *ACM SIGMOD 1979 International Conference on Management of Data, Proceedings*, pages 23–34. ACM Press, 1979.

[OW89] G. Ozsoyoglu and H. Wang. A relational calculus with set operators, its safety, and equivalent graphical languages. *IEEE Transactions on Software Engineering*, 15(9):1038–1052, 1989.

[SS86] H.-J. Schek and M.H. Scholl. The relational model with relation-valued attributes. *Information Systems*, 11(2):137–147, 1986.

[TF86] S. Thomas and P. Fischer. Nested relational structures. In P. Kanellakis, editor, *The Theory of Databases*, pages 269–307. JAI Press, 1986.

[Ull90] J. Ullman. *Principles of Database and Knowledge-Base Systems, volumes I and II.* Computer Science Press, 1989-1990.

[VdB91] J. Van den Bussche. Evaluation and optimization of complex object selections. In C. Delobel, M. Kifer, and Y. Masunaga, editors, *Deductive and Object-Oriented Databases*, volume 566 of *Lecture Notes in Computer Science*, pages 226–243. Springer-Verlag, 1991.