EQUIVALENCE OF THE RELATIONAL ALGEBRA AND
CALCULUS FOR NESTED RELATIONS

LUCY GARNETT and ABDULLAH U. TANSEL
Bernard M. Baruch College
17 Lexington Avenue, Box 513, New York, N.Y. 10010, U.S.A.

(Received December 1989 and in revised form February 1991)

Abstract—The relational model is extended to include nested structures. This extension is formalised using the distinction between a tuple scheme and a relation scheme. The algebra and calculus languages are defined for this model. It is shown that the restricted powerset is a derived operation in the algebra and the full powerset is expressible by a safe formula in the calculus. Since the full powerset cannot be derived from the algebra operations, there does not exist a complete equivalence between the calculus and the algebra. In other words, given any algebra expression, there is a safe calculus formula with equivalent expressive power. Conversely, given any safe calculus formula and a bound on the cardinality of the database instance, there is a corresponding equivalent algebra expression. The relational algebra is then augmented with programming constructs and this augmented algebra is shown to be equivalent in expressive power to the relational calculus for nested relations.

1. INTRODUCTION

Many applications using DBMS's require data structures to contain relations within relations. For example in a desktop publishing environment one manipulates page layouts. These page layouts themselves are broken into individual columns and each column can have a different makeup. Nested in the structure of a column can be a graphical object which itself is composed of various objects such as circles, rectangles, patterns, etc. Similar applications are also encountered in office automation, computer aided manufacturing, statistical databases, etc. One can generalize the relational data model to allow relations to occur as attribute values in tuples of relations. Such relations are called non first normal form (N1NF) relations or nested relations and have been the focus of recent research as contrasted with first normal form relations which can have nested relations.

In this paper we formalize such a model as a natural extension of the original relational model [1]. The basic objects being manipulated by the algebra are relations. However, the variables in the calculus language represent tuples. These two types of objects are related by the fact that the collection of tuples satisfying a calculus formula form a relation. We augment the concept of a scheme by defining two types of schemes, tuple and relation. A tuple scheme is an ordered sequence of previously defined schemes. Specification of a tuple consists of the arity of the tuple and the scheme of each of its components. The lowest level tuple scheme consists of indecomposable atoms. A relation instance is a set of tuples, all of one type, and its scheme is represented by specifying its tuple scheme. This inductive definition can be represented by a rooted tree. Although this definition might seem complicated it is necessary to distinguish between a tuple scheme and a relation scheme in order to be able to formalize the theory. For example, the membership operator in the calculus is only well defined when specifying that a tuple is a member of a relation made up of those tuple types.

Our formalism has several new aspects. Objects are classified according to a scheme. We assign to each scheme an ordinal called the order of the scheme. This numbering of the schemes allows us to give explicit inductive definitions and proofs. The definition of our typed tuple calculus is very similar to the standard definition. However, this new definition allows formulas which test
for membership in relations. The calculus requires that formulas not only be well formed but well typed. We define a relational algebra in much the same fashion as done by other researchers in the literature. The algebra includes formulas for membership and subset tests. We prove that these can be derived from the usual comparison operators. This algebra is powerful enough to derive a restricted form of the powerset operator which depends upon the particular interpretation, whereas the calculus language is powerful enough to include a single statement which expresses the full powerset of its argument independent of any interpretation. We then prove the strongest possible form of equivalence between the expressive power of the algebra and the calculus which is a new result to the best of our knowledge. More precisely, given any expression in the algebra, there is a safe calculus formula representing the same set as the algebra expression. Furthermore, given any safe formula in the calculus and any instance of the database, there is an expression in the algebra representing the same set as the calculus formula. We add more power to the relational algebra by augmenting it with programming constructs and show that this augmented algebra has the same expressive power as the relational calculus independent of the instances of the symbols mentioned in the expressions.

The rest of the paper is organized as follows. The next section contains a survey of previous work. In Section 3 the definition of a scheme and its order are given. Section 4 defines an instance of a scheme. The calculus is given in Section 5. Section 6 defines the interpretation of the calculus. Section 7 presents the relational algebra operators and how to construct powerful operators using the existing ones. The proof of the equivalence of the calculus and algebra (provided relation instances) comprises Section 8. Section 9 gives an example to illustrate the algorithms in the proof of equivalence of the calculus and algebra. Section 10 adds more power to the relational algebra and a proof of instance independent equivalence between this more powerful algebra and calculus. Formally we use a column index instead of an attribute name. However we feel free to use attribute names in our examples to improve readability.

2. PREVIOUS WORK

There is an explosion in the research on nested relations. For a long time research in relational database theory concentrated on normalized (flat) relations, as had been proposed by Codd [1]. Codd, himself, pointed out the usability of nested relations, but this path has not been pursued until recently. Makinouchi first proposed nested relations for non traditional applications, i.e., picture processing, etc. [2]. Jaeschke and Schek studied relations having set-valued attributes and proposed one-attribute nest and unnest operators [3]. Later Jaeschke defined recursive and non-recursive relational algebra languages for arbitrarily nested relations [4,5]. A recursive algebra for nested relations was also formulated by Schek and Scholl [6]. Fischer and Thomas defined a relational algebra for N1NF relations and generalized the nest and unnest operators to multi-attribute operators [7]. Kuper and Vardi [8] proposed a data model where the schemes are directed graphs. This permits cycles to occur. In order to handle this level of generality their theory distinguishes between l-values which constitute the space of addresses, and r-values which constitute the data space. Their algebra includes a powerset operator which permits them to prove its equivalence with their calculus language. Our schemes do not permit cycles and deal with only the data space. Our equivalence does not depend upon building the powerset operator into the algebra.

Roth, Korth and Silberschatz [9] extended the relational model for nested relations based on database logic formalism of Jacobs [10], defined a calculus language, and attempted to show the equivalence of this language to the algebra defined by Jaeschke and Schek and Thomas and Fischer [3,7]. However, they only consider nested relations which have at least one atomic attribute, and are in partitioned normal form [9]. It also includes extended definitions of algebra operations for nested relations. Roth, Korth and Silberschatz's article fails to prove the equivalence of the relational algebra and the relational calculus for nested relations. They give a method to translate from the relational calculus to an extended relational algebra having extended set operators which are based on the idea of combining (collapsing) tuples agreeing on their key (atomic) attributes. They also assume the convertibility of extended relational algebra expression to an equivalent relational algebra expression is sufficient reason to claim validity of their equivalence proof. However, the semantics of extended relational algebra is not the same as the semantics of
the standard relational algebra. For example, in going from the algebra to the calculus the union operator is translated into the logical disjunction operator. In translating from the calculus to the algebra, the disjunction becomes an extended union. But the union operator is not the same as the extended union operator. Furthermore, translating a calculus expression to an extended relational algebra expression creates anomalous interpretations of the calculus objects which goes against the standard rules of logic. Additionally, their calculus to algebra translation requires the operand relations to be in partitioned normal form. This is done in order to insure that the operators interact properly. Yet their algorithm creates relations for the domains of tuple variables and these relations are not in partitioned normal form. For a detailed discussion of these issues see [11]. Furthermore, there are significant differences between our work and Roth, Korth and Silberschatz’s work. They use extended set operations in translation from the calculus to the algebra in order to create the result of their limited domains. We use algebra operations and create a restricted form of the powerset operator. They need a keying operator to put relations in partitioned normal, whereas we do not need any keying operator at all.

Ozsoyoglu and Ozsoyoglu also studied set-valued relations and defined pack (nest), unpack (unnest) and aggregation-by-template operators [12]. They also showed the equivalence of algebra and calculus languages having aggregate functions for set-valued relations [13]. Their approach is different from ours in two respects. We allow arbitrary nested relations whereas they allow only one level of nesting with set-valued attributes. Secondly, they include aggregates and use Klug’s approach [14] whereas we do not include aggregates and mimic Ullman’s method [15].

Expressive power of the relational algebra is explored in [16-20]. Chandra and Harel defined a powerful query language which includes programming constructs and showed this language is as powerful as the least fixed point closure of relational algebra [17,18]. Van Gucht showed that the relational algebra having nest and unnest operators is BP-complete [21]. In another study, Gyssens and Van Gucht added programming constructs and a powerset operator to the relational algebra with nest, unnest operators and proved that the algebra and programming constructs have the same expressive power as the relational algebra with the powerset operator [19].

Abiteboul and Bidoit used nested relations for data organized hierarchically with respect to a key attribute [22]. In the format model, Hull and Yap developed a framework for combining relational and hierarchical models [23]. Normal forms for nested relations were explored by Ozsoyoglu and Yuan [24] and Roth and Korth [25]. Finally Kambayashi et al. [26], and Fischer and Van Gucht [27,28] studied dependencies and N1NF relations.

3. SCHEMES

A relation instance is a set of tuples. Each tuple has a fixed number of components. The components are either atoms or relations. A database schema consists of the rules for building these nested relations out of indecomposable atoms. Therefore we do not define separate attribute names having specific domains. The column index of a relation will be used in place of the attribute name. Although, as a convenience to the reader, we may use attribute names to illustrate examples. Relations are hierarchically defined and a rooted and ordered tree can be used to completely describe the scheme of a relation.

In the traditional relational model, a relation is a set of tuples of fixed arity. The theory implicitly uses that arity. For example, in defining the calculus, variables represent tuples of some fixed arity and to clarify the setting, sometimes the arity is explicitly indicated. In order to present a theory of nested, non first normal form relations (N1NF), we need a more comprehensive typing of objects. The first major classification of objects is into relations and tuples. From that bifurcation, we make further divisions. The formal definition is given inductively on the order of the scheme which in essence characterizes the depth of the nesting in the object. This enables us to use induction to formally present other definitions and proofs. A scheme is intuitively a recursive concept and rigorous recursive proofs have their basis in induction.

The definitions of the relation scheme and tuple scheme are interwoven. Atom is the basic undefined concept upon which the recursive definition is built. When constructing instances of a scheme, we will interpret atom as an element of U, the universe of simple indecomposable values. The definition is inductive on the order of the scheme. A tuple scheme is a finite ordered set of previously defined schemes which are either atoms, or relations of tuples of a lower order.
The order of a tuple scheme is the maximal order of any of its components. The basis of this inductive definition starts with atoms having order zero. A relation instance consists of a set of tuples of a fixed scheme. The relation scheme is characterized by its tuple scheme. The order of a relation scheme is one more than the maximal order of the scheme of any of its tuple components. Therefore only tuple schemes may have order zero. We use the keywords relation and tuple to distinguish between the two different kinds of schemes. It is necessary to make this distinction in order to define a calculus which incorporates the membership operator. An object may be an element of another object only if the first object has a tuple scheme and the second object has a relation scheme consisting of tuples of the same type as the first object.

### 3.1. Inductive Definition of Scheme and the Order of a Scheme

The format for a tuple scheme $t$ is $\text{tuple: } (t(1), \ldots, t(n))$. The scheme is named by the identifier $t$. The keyword tuple is used to give the type of the scheme and $t(i)$ is the scheme of the $i^{th}$ component of scheme $t$. An interpretation of $t$ is an $n$-tuple whose $i^{th}$ component is an interpretation of the scheme $t(i)$. The format for a relation scheme $s$ is $\text{relation: } t$. The scheme is named by the identifier $s$. The keyword relation specifies the type of the scheme. An interpretation of $s$ is a set of tuples having scheme $t$. We sometimes abbreviate the notation for a relation scheme and write $s$ has scheme relation: $(t(1), \ldots, t(n))$. If $t$ is tuple scheme and $i$ is an index between one and the arity of the tuple scheme, then $t(i)$ is an indexed identifier used to represent the scheme of the $i^{th}$ component of scheme $t$. A relation scheme may not be indexed.

A scheme $t$ has order zero if and only if it is a tuple scheme all of whose components are atoms. We use "atom" as the basic undefined term. There are two kinds of schemes that have order one. One possibility is that the scheme is a relation whose tuples have order zero; it is a scheme for a flat relation in standard terminology. The other possibility is that the scheme is a tuple such that any component of the tuple is either an atom or a relation scheme of order one; only atoms or flat relations can be components of an order one tuple. A similar formulation provides the inductive setting which interweaves the definitions of order $k$ tuple and relation schemes. There are two kinds of schemes which have order $k$. The first kind is a relation scheme whose tuples have order $k - 1$. The second kind is a tuple scheme whose components schemes have order $k$ or less and at least one component has order $k$. Since a component of a tuple scheme is either an atom or a relation, this definition is not circular. The order of a relation scheme is one more than the order of its constituent tuples and the order of a tuple scheme is the maximum of the order of its component schemes.

Corresponding to each relation scheme $s$ there is a unique ordered and rooted tree indicating how $s$ has inductively been formed from atoms. The root of the tree is $s$ and the root's children correspond to the scheme of its attributes. Each child is either an atom, in which case it is a leaf, or the child is a relation scheme having children of its own. The process continues and the tree grows downwards. The depth of the final resultant tree equals the order of the scheme at the root. Thus, every scheme has a uniquely defined order, although different schemes might possess the same order.

#### Example 3.1
Let's define a relation scheme $Q$ having order 2.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$</td>
<td>the scheme tuple: $(\text{atom}, \text{atom})$</td>
</tr>
<tr>
<td>$R$</td>
<td>the scheme relation: $t$</td>
</tr>
<tr>
<td>$s$</td>
<td>the scheme tuple: $(\text{atom}, R)$</td>
</tr>
<tr>
<td>$Q$</td>
<td>the scheme relation: $s$</td>
</tr>
</tbody>
</table>

Scheme $t$ has order zero. Schemes $R$ and $s$ each have order one. Scheme $Q$ has order two. An instance of this scheme is given in Figure 1. Scheme $Q$ can be written by using Knuth's linear notation for trees as follows [29]: $Q$ is the scheme relation: $(\text{atom}, \text{relation: } (\text{atom}, \text{atom}))$. The corresponding tree diagram is:

#### Example 3.2
The next example is a nested relation, Department, taken from Jaeschke [4]. Dno (a department’s identification number), Dmgr (department manager), Project (the projects
belonging to the department), Equipment (equipment used in the department) and Ddes (description of the department) are its attributes. The attribute Project is itself nested and contains the attributes Pname (project name), Plead (the project leader), Emp (information about the employees in the project) and Pdes (a description of that project). Emp and Equip are themselves nested relations which consist of Eno (employee number), Ename (employee name) and Ino (equipment number), Ides (equipment description), respectively. This relation is built out of the following schemes:

<table>
<thead>
<tr>
<th>Order</th>
<th>Scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Dno, Dmgr, Pname, Plead, Eno, Ename, Pdes, Ino, Ides, Ddes have scheme</td>
</tr>
<tr>
<td>1</td>
<td>Emp has scheme relation : (Eno, Ename)</td>
</tr>
<tr>
<td>1</td>
<td>Equip has scheme relation : (Ino, Ides)</td>
</tr>
<tr>
<td>2</td>
<td>Project has scheme relation : (Pname, Plead, Emp, Pdes)</td>
</tr>
<tr>
<td>3</td>
<td>Dept has scheme relation : (Dno, Dmgr, Project, Equip, Ddes)</td>
</tr>
</tbody>
</table>

Every interior node of the tree is a relation scheme. The leaves are all atoms. Although attribute names are used, it is only a convenience. The order position of a child is given a name to help the reader.

4. INSTANCE OF A SCHEME

We define a domain of interpretation of a scheme $q$ relative to a set $A$, of atoms, and denote this domain by $\text{Dom}_q(A)$. An instance of scheme $q$ is an element of $\text{Dom}_q(A)$. This definition is given inductively on the order of the scheme.

4.1. Definition of $\text{Dom}_q(A)$

Order zero scheme: In this case $q$ is tuple : $(atom, \ldots, atom)$, an $n$-tuple of atoms for some value of $n > 0$. Then,

$$\text{Dom}_q(A) = A \times \cdots \times A$$

(the $n$-fold product).

Order 1 scheme: There are two possibilities.

Case a. $q$ is relation: $(atom, \ldots, atom)$, a set of $n$-tuples of atoms. Then $\text{Dom}_q(A) = P(A \times \cdots \times A)$ (the power set of the $n$-fold product of $A$).

Case b. $q$ is tuple : $(q(1), \ldots, q(n))$. In this case $q(i)$ is either an atom or an order one relation scheme. Then,

$$\text{Dom}_q(A) = L_1 \times L_2 \times \cdots \times L_n,$$

for $1 \leq j \leq n$ where

$L_j = A$, if $q(j)$ is an atom or

$L_j = \text{Dom}_{q(j)}(A)$, if $q(j)$ is an order one relation.

An instance of an order one tuple is an $n$-tuple whose components are either atoms or order one relations.
Order \( k+1 \) scheme: Again there are two possibilities.

Case a. \( q \) is relation: \( t \), where \( t \) is a tuple scheme of order \( k \). Then,

\[
\text{Dom}_q(A) = P(\text{Dom}_t(A)).
\]

An instance of an order \( k+1 \) relation is a set of tuples of order \( k \).

Case b. \( q \) is tuple: \( (q(1), \ldots, q(n)) \). Each \( q(i) \) is either an atom or an order \( k \) relation. Then

\[
\begin{align*}
\text{Dom}_q(A) &= L_1 \times L_2 \times \cdots \times L_n, \\
L_i &= A, & \text{if } q(j) \text{ is an atom or} \\
L_i &= \text{Dom}_{q(j)}(A), & \text{if } q(j) \text{ is a relation}.
\end{align*}
\]

An instance of an order \( k+1 \) tuple is an \( n \)-tuple whose components are either atoms or relation instances of order \( k+1 \) or less, with at least one of the components having order \( k+1 \).

4.2. Definition of Instance

An instance of a relation scheme \( q \) relative to a set \( A \) is an element of \( \text{Dom}_q(A) \). A database schema is a collection of relation schemes, and an instance of a database schema consists of instances of the relation schemes in the database schema. An instance is not only the set of tuples but also has a scheme. In this way each distinct relation scheme \( s \) has a distinct empty set \( \phi_s \). We use relation to mean a relation scheme or its instance. The meaning is clear from the context. Figure 2 gives an instance of the relation scheme in Example 3.1.

\[
\begin{align*}
\{ & (a, \{(c,3),(e,5)\}), \\
& (a, \{(c,8),(f,2),(d,1)\}), \\
& (b, \{(c,8),(g,9)\})
\}
\end{align*}
\]

Figure 2.

Each one of the tuples in this set is an instance of tuple scheme \( s \). Note that a tuple scheme instance consists of one tuple, whereas a relation scheme instance consists of a set of tuples.

5. THE RELATIONAL CALCULUS

In the calculus language variables possess a tuple scheme. An indexed tuple variable refers to either a relation or an atom. Constants may have either a tuple or a relation scheme. Predicates always have a relation scheme.

5.1. Symbols

a. Predicate names: \( P, Q, S \)

For each relation scheme in the database schema we have a finite number of predicate names, one for each relation instance in the database schema.

b. Variables: \( s,t,u,v, \ldots \)

Corresponding to each tuple scheme, whether or not it is in the database schema, are a countable number of variables. If \( s \) is variable then \( g \) represents the scheme associated with \( s \). Variables may be indexed. If \( s \) is a variable then \( s[i] \) is an indexed variable for \( i \), a value between 1 and the arity of \( s \). Only one level of indexing is allowed, i.e., an indexed tuple variable may not be indexed again. If \( s[i] \) is an atom, then the scheme associated with the indexed variable \( s[i] \) is tuple: \( \langle \text{atom} \rangle \). Otherwise, \( s[i] \) inherits the scheme \( g(i) \). If a variable's scheme is a one-tuple whose only component is an atom, we call it an atomic variable. Although a variable can only have a tuple scheme, an indexed variable has either a relation scheme or is atomic. Whenever the context is clear we omit the underlining.
C. Constants: a, b, c, ...

There are at most a countable number of constant symbols. Each constant has a scheme, either tuple or relation. As in the case of variables, underlining is used to denote the associated scheme.

5.2. Well Formed and Well Typed Formulas:

The set of well formed and well typed formulas are defined as follows:

Atomic Formulas (no logical operators)

1. \( P(t) \)
   - \( P \) is a predicate name and \( t \) is a variable. \( P \) has scheme relation: \( t \).
2. \( s[i] \text{op} t[j], s[i] \text{op} c, c \text{op} s[i], \)
   - where \( \text{op} \) is one of \(<, >, \leq, \geq \) and \( s[i], t[j] \) and \( c \) are atomic.
3. \( s[i] = t[j], s[i] = c, c = s[i], \)
   - where \( s[i], t[j], \) and \( c \) have the same scheme, which may or may not be atomic.
4. \( s \in t[j], \)
   - where \( s \) is a variable or constant and \( t[j] \) is an indexed variable with scheme relation: \( s \).
   - \( s \in c, \)
   - where \( s \) is a variable or constant and \( c \) is a constant with scheme relation: \( s \).
   - \( s[i] \in t[j], \)
   - where \( s[i] \) is atomic and \( t[j] \) is an indexed variable with scheme relation: \( \langle \text{atom} \rangle \).
   - \( s[i] \in c, \)
   - where \( s[i] \) is atomic and \( c \) is a constant with scheme: relation: \( \langle \text{atom} \rangle \).

Formulas Containing Logical Operators

5. If \( \psi \) and \( \lambda \) are formulas, then so are \( \psi \land \lambda, \psi \lor \lambda, \) and \( \neg \psi. \)
6. If \( \psi \) is a formula with a free variable \( t, \) then \( \exists t \psi (t) \) and \( \forall t \psi (t) \) are both formulas and \( t \) no longer occurs free in \( \exists t \psi (t). \)
7. \( r[i] = \{ s | \psi(s, t, u, v, \ldots) \} \) is a formula if \( \psi(s, t, u, v, \ldots) \) is a formula with free variables \( s, t, u, v, \ldots, \) and \( r \) doesn't occur free in \( \psi. \) The indexed variable \( r[i] \) has scheme relation: \( s. \) In the resulting formula, the variables \( r, t, u, v, \ldots, \) are free and \( s \) is bound. This formula is called the set formator.

We include this last formula because we need a way to build relations. We could have eliminated this and replaced it with a formula of the form \( \forall s (s \in r[i]) \leftrightarrow \psi(s, t, u, v, \ldots). \) However, this raises problems with the definition of safety, since safety is harder to verify when the universal quantifier \( \forall \) is in the formula. The form of this set building formula is due to Roth, Korth and Silberschatz [9].

6. INTERPRETATION OF THE CALCULUS LANGUAGE

The interpretation of the language is relative to a universe of atoms, \( U. \) An interpretation of a language gives an assignment of values to the elements in the language.

Interpretation of Constants

An interpretation \( I_c \) of each constant \( c \) is a value \( I_c \in \text{Dom}_c(U). \)

Interpretation of Predicate Names

An interpretation \( I_P \) of the predicate name \( P \) in the language is an instance \( I_P \) of the relation scheme \( P \) relative to the universe. In other words, \( I_P \in \text{Dom}_P(U). \)

Interpretation of Formulas

An interpretation of a formula is an interpretation of each of its constants and predicate symbols as well as an assignment to each of its free variables \( t, \) a value \( I_t \in \text{Dom}_t(U). \) \( I_t \) is a tuple and we write \( I_t(i) \) to denote the \( i^{th} \) component of \( I_t. \)
6.1. Satisfiability of Formulas

Step 0: Formulas with No Logical Operators.

1. \( P(t) \) is satisfied if \( I_t \in I_p \).

2. a. \( s[i] \text{ op } t[j] \) is satisfied if \( I_s(i) \text{ op } I_t(j) \).
   b. \( s[i] \text{ op } c \) is satisfied if \( I_s(i) \text{ op } I_c \).
   c. \( c \text{ op } s[i] \) is satisfied if \( I_c \text{ op } I_s(i) \),
      where \( \text{ op } \) is one of \( <, >, \leq, \geq, = \).

3. a. \( s \in t[j] \) is satisfied if \( I_s \in I_t(j) \).
   b. \( s \in c \) is satisfied if \( I_s \in I_c \).
   c. \( s[i] \in c \) is satisfied if \( I_s(i) \in I_t(c) \).
   d. \( s[i] \in c \) is satisfied if \( I_s(i) \in I_c \).

Step k+1: Formulas with \( k + 1 \) Logical Operators.

Assume satisfiability has been defined for all formulas with \( k \) or fewer operators, then satisfiability of a formula with \( k + 1 \) operators is defined by:

4. If \( \psi \) and \( \lambda \) each have \( k \) or fewer operators, then
   a. \( \psi \land \lambda \) is satisfied iff both \( \psi \) and \( \lambda \) are satisfied.
   b. \( \psi \lor \lambda \) is satisfied iff either \( \psi \) or \( \lambda \) is satisfied.
   c. \( \neg \psi \) is satisfied iff \( \psi \) is not satisfied.

5. If \( \psi \) is a formula with \( k \) operators and a free variable \( t \), then
   a. \( \exists t \psi(t) \) is satisfied iff there is some assignment to the variable \( t \) which satisfies \( y(t) \).
   b. \( \forall t \psi(t) \) is satisfied iff for any assignment to the variable \( t \), satisfies \( y(t) \).

6. Let \( \psi \) be a formula with \( k \) operators; then the set building formula \( r[i] = \{ t \mid \psi(s, t, u, v, \ldots) \} \)
   is satisfied by the interpretations \( I_s, I_t, I_u \) and \( I_v \) of its free variables, if the following condition is met: \( I_r(i) \) equals the set of assignments \( \{ I_s \} \) satisfying \( \psi(s, t, u, v, \ldots) \) for the interpretations \( I_t, I_u \) and \( I_v \). If there are no such tuples \( I_s \) and \( I_r(i) \) is empty, then we say that this set building formula is not satisfied. Nonemptiness is a crucial requirement because in a later section we use this formula to express the relational algebra's nesting operation. If there is nothing to nest, then the algebra's nesting operation won't produce any tuples. In that case, we don't want the calculus to produce a tuple having the empty set for a nested component.

For notational convenience, if \( \psi(s, t, u) \) is a formula with free variables \( s, t, u \), then \( \{ (s, t, u) \mid \psi(s, t, u) \} \) denotes the set of tuples \( (I_s, I_t, I_u) \) satisfying \( \psi(s, t, u) \), where \( I_s, I_t, I_u \) is the concatenation of the values \( I_s, I_t, I_u \).

The calculus language, as defined above, allows formulas with infinite evaluations. For example, the formula \( t = t \) will be satisfied by every element in the universe \( U \), and yet from a practical standpoint it might not be possible to determine all the elements of \( U \). In fact, given a database, supplied with a fixed number of relations which are viewed as interpretations of the predicate symbols, we are interested in queries which can only be answered relative to these relations. Therefore we must build that limitation into the language. We do this by defining a concept of a safe formula, modeled after Ullman's definition [15].

6.2. Definition of the Domain of Interpretation

Given a universe \( U \) of atomic values, a scheme \( t \) (tuple or relation), and an instance \( I_t \) of that scheme, i.e., an element of \( \text{Dom}_t(U) \), \( \text{ATOMS}(I_t) \) is defined to be the set of all the atoms needed to form \( I_t \). More precisely, \( \text{ATOMS}(I_t) \) equals the smallest subset \( A_t \) of \( U \) such that \( I_t \in \text{Dom}_t(A_t) \). \( \text{ATOMS}(t) \) is used as an abbreviation for \( \text{ATOMS}(I_t) \). If \( \psi \) is a formula in the relational calculus language, then given an interpretation of the language, \( \text{ATOMS}(\psi) \) is defined to equal \( \text{ATOMS}(P_1) \cup \ldots \cup \text{ATOMS}(P_n) \cup \text{ATOMS}(c_1) \cup \ldots \cup \text{ATOMS}(c_m) \). This union takes place over all predicate symbols \( P \) and constants \( c \) occurring in \( \psi \). \( \text{Dom}(\psi) \) equals
Dom$_t$(ATOMS($\psi$)). This set contains all possible objects of scheme $t$ built up from the atoms of predicates occurring in $\psi$. If the context is clear we let $D_t$ denote Dom$_t$(ψ).

The safety of formula $\psi$ with free variable $t$ is defined relative to an interpretation of the predicate names and constant symbols in the formula. We define $\psi$ to be a safe formula when the following three conditions are satisfied:

1. \{ $t$ | $\psi(t)$ \} is a subset of Dom$_t$(ψ). This means that every interpretation of $t$ satisfying $\psi$ is built out of the atoms occurring in the interpretation of the predicate names and constants occurring in $\psi$.
2. If $\exists u (\omega(u, v, \ldots, ))$ is a subformula of $\psi$, then any interpretation of $u$ which satisfies $\omega$ is in Dom$_u$(ω), no matter how the other free variables are interpreted.
3. If $\forall u (\omega(u, v, \ldots, ))$ is a subformula of $\psi$, then any interpretation of $u$ not in Dom$_u$(ω) satisfies $\omega$, no matter how the other free variables are interpreted.

This definition of safety is a direct extension of the one given by Ullman [15]. The next theorem gives a safe formula expressing the powerset.

**Theorem 6.1.** Given any safe formula $\psi(t)$ with one free variable, there is a safe formula $\omega(s)$ with one free variable such that \{ $s$ | $\omega(s)$ \} equals the powerset of \{ $t$ | $\psi(t)$ \}.

**Proof.** Let $\psi(t)$ be a safe formula with free variable $t$. Let $\omega(s)$ be the formula $\exists u (s(1) = u(1) \land s(1) = \{ t | t \in u(1) \land \psi(t) \})$. By definition, any variable in the calculus has a tuple scheme. However, we want the variable $s$ to represent an element of the powerset. We do this by making $s$ a tuple variable with one component. That component is a relation whose tuples have scheme $t$. For example, if $t$ has scheme tuple : (atom, atom, atom) then $s$ has scheme tuple : (relation : (atom, atom, atom)). The variable $u$ has the same scheme as $s$ and is needed in order to insure that $\omega(s)$ is a safe formula. Although $u$ ranges over all possible tuples in the universe, the rest of the formula places a limit on the possible values of $u$ that can satisfy $\omega$ and thus insuring the safety of $\omega$. It is easy to check that a 1-tuple satisfies $\omega$ if and only if its unique component is a subset of tuples satisfying $\psi$.

7. THE RELATIONAL ALGEBRA

The relational algebra is a collection of expressions. The operands of a relational algebra expression are either constant relations or variables representing a relation. The operations are union, intersection, difference, Cartesian product, projection, selection, nesting and unnesting. The standard derived operations such as intersection, join, division, etc., can be expressed in terms of these basic operations. A relational algebra expression is essentially a function. It accepts relations as arguments and outputs a computed relation. For notational convenience, if $E$ is a relational algebra expression then $EV(E)$ denotes the computed relation. We say that $E$ expresses the relation $EV(E)$. Relation variables in algebra expressions correspond to predicate symbols in the calculus language.

7.1. Definition of the Algebraic Operations

$E$, $R$, and $S$ denote algebraic expressions.

1. **Union:** $R \cup S$.
   
   $R$ and $S$ have the same scheme. $R \cup S$ has that scheme and $EV(R \cup S)$ equals $EV(R) \cup EV(S)$, the standard set union.

2. **Intersection:** $R \cap S$.
   
   $R$ and $S$ have the same scheme. $R \cap S$ has that scheme and $EV(R \cap S)$ equals $EV(R) \cap EV(S)$, the standard set intersection.

3. **Set difference:** $R - S$.
   
   $R$ and $S$ have the same scheme. $R - S$ has that scheme and $EV(R - S)$ equals $EV(R) - EV(S)$.

4. **Cartesian product:** $E = R \times S$.
   
   $R$ has scheme relation : ($r(1), \ldots, r(n)$).
   
   $S$ has scheme relation : ($s(1), \ldots, s(m)$).
   
   $E$ has scheme relation : ($e(1), \ldots, e(n + m)$), where $e(i) = r(i)$ for $1 \leq i \leq n$ and $e(i + n) = s(i)$ for $1 \leq i \leq m$. $EV(E)$ equals $EV(R) \times EV(S)$, the standard set Cartesian product.
5. Projection: $E = \pi_Y(R)$.
   
   $R$ has scheme relation: $\langle r(1), \ldots, r(n) \rangle$.
   
   $Y = \{i_1, \ldots, i_k\}$ is a subset of the indexes $\{1, \ldots, n\}$.
   
   $E$ has scheme relation: $\langle e(1), \ldots, e(k) \rangle$, where $e(s) = r(i_s)$ for $1 \leq s \leq k$.
   
   $EV(E) = \{(x_{i_1}, \ldots, x_{i_k}) \mid \exists (y_{i_1}, \ldots, y_{i_k}) \in EV(R) \text{ and } x_s = y_{i_s}, \text{ for } 1 \leq s \leq k\}$, which can also be written as $\{t[Y] \mid t \in EV(R)\}$.

6. Selection: $E = \sigma_F(R)$.
   
   $R$ has scheme relation: $\langle r(1), \ldots, r(n) \rangle$.
   
   $E$ has scheme relation: $\langle r(1), \ldots, r(n) \rangle$.

$F$ is a formula which selects tuples in the relation $EV(R)$ to form the relation $EV(E)$. In defining admissible formulas we could be very generous and permit many formulas, instead we keep our formulas simple. However, we show that more complicated selection formulas can be derived from the simple ones.

The format of $F$ is $iopj$ where $op$ is one of the elementary comparison operators: $=, <, >, \neq, \leq, \geq$. The operands $i$ and $j$ can either be an index to a tuple component in the relation or constant value. We use an apostrophe to distinguish a constant value from an index. The schemes for $i$ and $j$ must be the same. Furthermore, if $op$ is neither $=$ nor $\neq$, then $r(i)$ and $r(j)$ must both be atoms. If we view $R$ as a tree structure, then $i$ and $j$ are either constants or refer to the children of the root since we only allow indexing on one level.

From this restricted class of formulas we can derive a much larger class. In what follows $k, k_1, \ldots, k_s$ and $h$ are constants or indexes to columns of the relation; that is they represent children of the root of the relation $R$. The following are the derived selection formulas:

6a. $k \in h$.
   
   $r_k$ has scheme tuple : $\langle \text{atom} \rangle$, and
   
   $r_h$ has scheme relation : $\langle \text{atom} \rangle$.

6b. $(k_1, \ldots, k_s) \in h$.
   
   $h$ has scheme relation : $\langle r(k_1), \ldots, r(k_s) \rangle$.

6c. $h \supseteq k$.
   
   $h$ and $k$ have the same relation scheme.

The second case permits grouping of constants or columns of the relation and treating this group as a tuple, to ask if that tuple is a member of a relation occurring either as a constant relation, or as another column of the original relation. The third case asks if the component relation in column $k$ is a subset of the component relation in column $h$. The proof of these derived operations is deferred to the end of the definition. See Lemma 7.3.

**Example 7.1.** $R$ is a scheme consisting of 2-tuples. Each column of the 2-tuple is a set of atoms. $R = \text{relation} : \langle \text{relation} : \text{atom}, \text{relation} : \text{atom} \rangle$. An instance of $R$ is:

\[
\{ \{a, c, d\}, \{a, b, c, d\} \\
\{b, c, d\}, \{a, b\} \\
\{e, f, g\}, \{e, f, h, d\} \\
\{a, c, d\}, \{a, c, d\} \}
\]

If $F$ is the formula `$b \in 1$ then $\sigma_F(R)$ produces:

\[
\{ \{b, c, d\}, \{a, b\} \}
\]

If $F$ is the formula `$c \supseteq 1$ then $\sigma_F(R)$ produces:
Relational algebra and calculus

{ { {a,c,d}, {a,b,c,d} } 
  { {a,c,d}, {a,c,d} } }

Selection formulas with logical connectives can be derived from this simple selection format. The conjunction of two formulas is the consecutive application of selection by the formulas. The disjunction of two formulas is the union of their selection. Negation of a formula is handled by pushing the negation into the atomic components of the formula and noting that the negation of any $op$ is also an $op$. We are also able to derive an operation that performs selection using the membership operator on deeply nested structures. See Lemma 7.4 at the end of this section.

7. Nesting: $E = \forall Y(R)$

$R$ has scheme relation: $(r(1),...,r(n))$.
$Y = \{i_1,...,i_k\}$ is a subset of the indexes $\{1,...,n\}$.
$X = \{h_1,...,h_{n-k}\} = \{1,...,n\} - Y$ (i.e., $X$ is the complement of $Y$).
$E$ has scheme relation: $(e(1),...,e(n-k+1))$, where
$c(j) = r(h_j)$ for $1 \leq j \leq n-k$ and
c($n-k+1$) has scheme relation: $(r(i_1),...,r(i_k))$.

$EV(E) = \{(z_1,...,z_{n-k+1}) | \exists(s_1,...,s_n) \in EV(R),\ and\ x_j = s_{h_j}\ for\ 1 \leq j \leq n-k,\ and\ z_{n-k+1} = \{(y_1,...,y_k) | \exists(v_1,...,v_n) \in EV(R),\ and\ v_{h_j} = x_j,\ for\ 1 \leq j \leq n-k,\ and\ v_{i_j} = y_j,\ for\ 1 \leq j \leq k}\}.$

$EV(E)$ consists of tuples in $EV(R)$ with the $Y$ components nested together if they agree on the indexes in $X$. In this definition, the nested component never is empty, at the very minimum it contains the tuple $(s_{i_1},...,s_{i_k})$.

8. Unnesting: $E = \mu_k(R)$.

$R$ has scheme relation: $(r(1),...,r(n))$.
$r(k)$ has scheme relation: $(t(1),...,t(m))$.
$E$ has scheme relation: $(e(1),...,e(n+m-1))$, where
c($i$) = $r(i)$, for $1 \leq i \leq k-1$, and
c($i$) = $r(i+1)$, for $k \leq i \leq n-1$, and
c($i$) = $t(i-n+1)$, for $n \leq i \leq n+m-1$.

$EV(E) = \{(x_1,...,x_{n+m-1}) | \exists(s_1,...,s_n) \in EV(R)\ and\ \exists(y_1,...,y_m) \in s_k\ and\ x_i = s_i,\ for\ 1 \leq i \leq k-1,\ and\ x_i = s_{i+1},\ for\ k \leq i \leq n-1,\ and\ x_i = y_{i-n+1},\ for\ n \leq i \leq n+m-1\}.$

$EV(E)$ consists of tuples whose first $n-1$ components agree with all the components, except the $k$th, of some tuple in $EV(R)$ and whose last $m$ components are in the $k$th component of that tuple in $EV(R)$.

Although the relational algebra operations might look limited in scope, in fact they are very powerful. The standard operations such as join, intersection, and quotient can be derived in the usual way. We derive many more operations which are presented in the following lemmas.

Given any integer $k \geq 0$, define the operator $P_k$ such that for any algebraic expression $E$ and any evaluation function $EV$ of the base variables, $EV(P_k(E))$ equals those subsets of the $EV(E)$ with $k$ or fewer elements. We call $P_k$ the $k$-restricted power set operator.

**Lemma 7.1.** $P_k$ is expressible in the relational algebra.

**Proof.** $E$ is an expression in the relational algebra having arity $n$. Let $Y$ equal $\{nk+1,\ldots,nk+n\}$. Then $P_k(E) = (\forall_{nk+1} \nu_Y \sigma_F(E^{k+1}))) \cup \{\emptyset\}$ where $E^{k+1}$ is the $k+1$ fold product of $E$ having arity $n(k+1)$. $F$ is the statement selecting the tuples whose last copy of $E$ agrees with one of the first $k$ copies. More precisely, $F$ is the formula $F_1 \land \cdots \land F_j \land \cdots \land F_k$, where $F_j$ states that the $j$th copy agrees with the $(k+1)^{st}$ copy. Formally we have, $F_j = F_{ij} \land \cdots \land F_{ij} \land \cdots \land F_{nj}$ for $1 \leq j \leq k$, where $F_{ij}$ states that the $i$th component of the $j$th copy agrees with the $i$th component of the $(k+1)^{st}$ copy. $F_{ij}$ is the formula $nk+i = (j-1)n+i$ for $1 \leq i \leq n$. The operator $\nu_Y$ nests
the last $n$ components of the selected set forming the $k$-element or less subsets of $E$. Projection \( \pi_{k,n+1} \) onto the last (nested) column produces the desired result. Only the empty set is missing and this can be added with the union operator. Note that when $k > n$, where $n$ is the cardinality of \( EV(E) \), the $k$-restricted power set operator still makes sense and produces all the subsets of $EV(E)$.\]

**Example 7.2.** Let $R$ be a relation having scheme relation: (atom). An instance of $R$ is \{$(a), (b)$\}. In our proof $k$ equals 2 and $n$ equals 1. We form the 3-fold product and perform the selection, which requires that the third component equal one of the first two. This produces:

\[
\begin{array}{c}
\{(a\ a\ a),
(a\ b\ a),
(a\ b\ b),
(b\ a\ a),
(b\ a\ b),
(b\ b\ b)\}
\end{array}
\]

After nesting on the last factor we obtain:

\[
\begin{array}{c}
\{(a\ \{a\}),
(a\ b\ \{a, b\}),
(b\ a\ \{a, b\}),
(b\ b\ \{b\})
\end{array}
\]

Projection on the last factor gives the relation

\[
\{\{(a)\},\{(a, b)\},\{\{b\}\}\}.
\]

The last step adds the empty set to produce

\[
\{\{(a)\},\{(a, b)\},\{\{b\}\},\{\{\}\}\}.
\]

**Corollary 7.1.** If $E$ is an expression such that $EV(E) = X$, then there is an expression $E'$ such that $EV(E') = P(X)$, the full power set of $X$.

**Proof.** Let $k$ equal to the number of tuples in $X$ and apply the above lemma. As seen in Lemma 7.1, the $k$-restricted powerset operator, $P_k$, does not depend on the instance of its operand relation, and the corresponding algebraic expression works for all the possible instances of the operand relation. It has been shown in [19] that the full powerset can not be expressed in the relational algebra. The powerset operator grows exponentially whereas an operator in the algebra has only polynomial growth. On the other hand, the $k$-restricted powerset of a relation is equal to the full powerset of that relation for any $k$ greater than the cardinality of that relation. New programming constructs, such as the while loop, or an operator for the powerset can be added to the relational algebra [19]. We consider this extension to the relational algebra in Section 10.

**Corollary 7.2.** If $X$ is expressible in the algebra by $E$, then the set of all $k$ elements subsets is expressible in the algebra by $P_k(E) - (P_{k-1}(E))$.

**Proof.** Straightfoward from the definition. Diagonalization is a useful operator. Let $E$ be a expression in the algebra whose scheme has arity $n$. Then the diagonal of $E$ is the expression given by:

\[
\text{Diag}(E) = \{t \in EV(E) \times EV(E) \mid t[1] = t[n+1] \land \cdots \land t[i] = t[n+i] \land \cdots \land t[n] = t[n+n]\}.
\]
Additionally, the $i^{th}$ diagonal of $E$ defined for every $i$ between 1 and $n$ is given by:

$$\text{Diag}_i(E) = \{ t \mid t \in EV(E) \times \pi_i(EV(E)) \mid t[i] = t[n+1] \}.$$  

Diagonalization effectively creates a key for a relation. One of the two copies functions as a key. This idea has been used in [12,14].

**Lemma 7.2.** The diagonalization operators are in the relational algebra.

**Proof.** The full diagonal is a result of selection applied to the Cartesian product of $E$ with itself. The $i^{th}$ diagonal results from selection applied to the Cartesian product of $E$ with the projection of $E$ onto its $i^{th}$ column.

**Lemma 7.3.** The derived forms of the selection operator $\sigma_F$ can be expressed in terms of the elementary comparison operators; $\neq, \geq, <, \leq$.

**Proof.** $R$ has scheme relation: $(r(1), \ldots, r(n))$ and the formula $F$ is one of the following:

- a. $k \in h$.
  
  $r(k)$ has scheme tuple: $(\text{atom})$, and
  
  $r(h)$ has scheme relation: $(\text{atom})$.

- b. $(k_1, \ldots, k_s) \in h$.
  
  $h$ has scheme relation: $(r(k_1), \ldots, r(k_s))$.

- c. $h \supset k$.
  
  $h$ and $k$ have the same relation scheme.

Case a. Let $W = \text{Diag}_h(R)$. Unnest the last column of $W$. Select those tuples whose $k^{th}$ column equals this unneasted last column. Then eliminate the last column by projecting onto the first $n$ columns.

Case b. Let $W = \text{Diag}_k(R)$. Unnest $W$ over its $(n+1)^{th}$ column to produce a relation with arity $s+n$. The selection operation is iterated $n$ times, with $F_i$ equal to $(k_i = n+i)$ for $1 \leq i \leq n$. Each formula checks to see that the $k_i$ column equals the $i^{th}$ column of the unnested $h$. The final step is projection onto the first $n$ columns to express the correct subset of the original relation.

Case c. Suppose $r(h)$ and $r(k)$ are relation schemes whose tuples have arity $m$. We form $W = \text{Diag}_k(R)$. $W$ has arity $n+1$. Unnest $W$ along this second copy of $k$, that is $V = \mu_{n+1}(W)$. $V$ has arity $n+m$. Then use Case b of this lemma, i.e., the derived selection operation $\sigma_{F}(W)$, where $F$ is of the form $(n+1, \ldots, n+m) \in h$. Afterwards, nest on the last $m$ columns to reform the second copy of $k$ and compare it to the original copy of $k$, using selection with the formula $F$, $k = n+1$. All those tuples whose $k^{th}$ column was not a subset of the $h^{th}$ column will be eliminated. The final step is projection onto the first $n$ columns to obtain a subset of the original relation. This proof has one flaw. If column $k$ of a tuple in $R$ is empty, then the selection operation should keep that tuple, since the empty set is a subset of any set. However, when we unnest the second copy of column $k$, that entire tuple is eliminated. We can overcome that difficulty by first using selection to obtain $R' = \sigma_{F'}(R)$ where $F'$ is $s = \phi_{r(k)}$ and $\phi_{r(k)}$ is the empty set having scheme $r(k)$. In other words $R'$ is the subset of $R$ whose $k^{th}$ column is empty. The proof now proceeds as before, but the last step is to add $R'$ back to the result using the union operation. Note that the empty set was not a problem in Case a of this lemma, because nothing is an element of the empty set. It requires treatment in Case c because the empty set is a subset of any set.

The negation of all these operations can also be derived. For example, if $F$ is $k \in h$ and $F'$ is $k \not\in h$, then $\sigma_{F'}(E) = E - \sigma_F(E)$.

The selection operators and derived selection operators only query the children of the root of the relation. However, we can generalize the selection operator to handle queries of any nodes in the tree structure of a relation scheme. If $E$ expresses a relation of order $n$ and $G$ and $H$ are nested relations in it, we can use selection formulas of the form $G \oplus H$ where $\oplus$ is one of $\in, =, <, >$, etc., and is consistent with the type of the operands.

**Lemma 7.4.** Selection can be done on deeply nested structures.
**Proof.** An outline of the proof follows. The specification of the nested substructures $G$ and $H$ are given by the branch path from the root to the nodes representing them.

1. Form $\text{Diag}(E)$: The second factor of $E$ is a scratch pad to unnest and perform the appropriate selection, and the first factor of $E$ acts as the key to the tuple.
2. Apply unnesting as many times as necessary to the second copy of $E$ to make both $G$ and $H$ structures flat. This is similar to a branch unnest operation as described in [5]. Note that the order and number of these unnest operations do not matter [7].
3. These flattened versions of $G$ and $H$ are now the children of the root and a query of the type in Lemma 7.3 can be applied.
4. After the weeding by selection is done, project onto the first factor of $E$.

This proof is rather sketchy and leaves out an important case. What happens if in the process of following Step 2, an empty relation is encountered? In our theory, the entire tuple will be eliminated, because unnesting an empty set produces no tuples. If this is what is intended, no problem arises. However, in certain cases, such tuples may satisfy the conditions of the query and need to be retained. The proof of this lemma can be extended to produce a result of non-elimination. However, an argument could be made on the side of elimination. We don’t get into these issues here, but rather state that this lemma is not needed for any other proofs in this paper.

Our proof here is different than the approaches provided in the literature [4,9]. They flatten the original relation first and then do the selection on the flat relation. This is followed by a series of nest operations to reconstruct the original scheme. It is well known that nesting is not the inverse operation of unnesting in all cases. Hence their approach does not always produce the original relation. On the other hand, our method always results in the original relation. This is accomplished by forming a larger relation whose factors are the original relation together with a scratch copy which is later eliminated.

**Example 7.3.** Referring to the relation $Q$ given by Figure 2, we find all tuples in $Q$ whose $tt$ column contains a tuple whose second column is less than 5. Following the ideas in Lemma 7.4, we obtain the following algebraic expression:

$$\pi_{1,2} \sigma_{s < '5'} \mu_4 \text{Diag}(Q),$$

which produces the relation

$$\{ \begin{array}{c} a, \{(c, 3), (e, 5)\} \\ a, \{(c, 5), (f, 2), (d, 1)\} \end{array} \}$$

Forming the diagonal in the first step, serves as keying the tuples. If we had flattened the relation first and used the expression $\nu_{(2, 3)} \sigma_{s < '5'} \mu_2(Q)$, we would produce the relation

$$\{ \begin{array}{c} a, \{(c, 3), (f, 2), (d, 1)\} \end{array} \}.$$
previous section. To pave the way for this equivalence proof we now show that the sets needed to express safety are expressible in the relational algebra.

**Lemma 7.5.** If $I_P$ is an interpretation for a predicate name $P$, then $\text{ATOMS}(I_P)$ is expressible in the relational algebra. The same holds true for interpretations of the constant relations $[7]$.  

**Proof.** Flatten the relation $P$ by repeated application of the unnest operation. Then project on each column and take the union of the projections.  

**Lemma 7.6.** If $\psi$ is a formula in relational calculus, then $\text{ATOMS}(\psi)$ is expressible in the relational algebra.  

**Proof.** Take the union of the $\text{ATOMS}(I_P)$ for each $P$ occurring in $\psi$ together with the union of the $\text{ATOMS}(I_c)$ for each constant $c$ occurring in $\psi$.  

**Lemma 7.7.** If $A$ is a set expressible in the relational algebra then $\text{Dom}_t(A)$ is expressible in the relational algebra for any scheme $t$, be it tuple or relation. In other words if there is an expression $E$ and an evaluation $EV$ such that $EV(E) = A$, then there is an expression $E_t(A)$ such that $EV(E_t(A)) = \text{Dom}_t(A)$.  

**Proof.** The proof is by induction on the order of $t$ and the fact that $\text{Dom}_t(A)$ equals the $k$-restricted power set, where $k$ is the cardinality of $A$.

**Order Zero:**

$t$ has a scheme, tuple: $(\text{atom}, \ldots, \text{atom})$ with arity $n$. Then $\text{Dom}_t(A)$ is the $n$-fold Cartesian product of $A$.

**Order One:**

Case a: $t$ has scheme relation: $(\text{atom}, \ldots, \text{atom})$ with arity $n$. By definition $\text{Dom}_t(A) = P(A \times \cdots \times A)$ (the $n$-fold product). Let $k$ equal the cardinality of $A \times \cdots \times A$. Then $P(A \times \cdots \times A)$ equals $P_k(A \times \cdots \times A)$ which is expressible in the algebra.

Case b: $t$ has scheme tuple: $(t(1), \ldots, t(n))$, where each $t(i)$ is either an atom or a relation scheme of order 1. Then $\text{Dom}_t(A) = L_1 \times L_2 \times \cdots \times L_n$, where for each $j, 1 \leq j \leq n$,

(i) $L_j = A$, if $t(j)$ is an atom, or  
(ii) $L_j = \text{Dom}_{t(j)}(A)$, if $t(j)$ is a relation of order zero tuples.

Subcase (ii) is an example of Case a, and $L_j$ is expressible in the relational algebra.

**Order $k+1$:**

Case a: $t$ has scheme relation: $s$, where $s$ is an tuple scheme having order $k$. By the induction hypothesis, $\text{Dom}_s(A)$ is expressible in the relational algebra. $P(\text{Dom}_s(A))$ equals $P_k(\text{Dom}_s(A))$ where $k$ is the cardinality of $\text{Dom}_s(A)$ and is expressible in the algebra. Hence, $\text{Dom}_t(A) = P(\text{Dom}_s(A))$.

Case b: $t$ has scheme tuple: $(t(1), \ldots, t(n))$, where $t(i)$ is either an atom or a relation scheme whose order is less than or equal to $k+1$. Then $\text{Dom}_t(A) = L_1 \times L_2 \times \cdots \times L_n$, where for each $j, 1 \leq j \leq n$,

(i) $L_j = A$, if $t(j)$ is an atom, and  
(ii) $L_j = \text{Dom}_{t(j)}(A)$, if $t(j)$ is a relation.

Subcase (ii) has already been done in Case a. Therefore, all the $L$'s are expressible in the relational algebra and so is their Cartesian product.  

Since the restricted power set operation is used, this proof depends upon the cardinality of the set $A$, or more precisely the cardinality of an instance of $A$. In Section 10, we give a procedure to generalize the algebra, so that this method works for any instance of $A$.  

LEMMA 7.8. Given a formula $\psi$ having any number of free variables, a tuple scheme $t$, and an interpretation $I$ of the predicate symbols occurring in $\psi$, then $\text{Dom}_t(\psi)$, as defined in section 5, is expressible in the relational algebra. In other words, there is an algebraic expression $E_{\psi,t,I}$ such that $\text{EV}(E_{\psi,t,I}) = \text{Dom}_t(\psi)$.

PROOF. $\text{Dom}_t(\psi) = \text{Dom}_t(\text{ATOMS}(\psi))$. Let $A = \text{ATOMS}(\psi)$ and apply Lemma 7.7. Once the interpretation of the base relations is fixed, we can determine the cardinality of the set of the atoms occurring in relations used in $\psi$. The restricted powerset operator accomplishes the desired result.

EXAMPLE 7.5. A safe formula expressing the query in example 7.4 is $\exists s(Q(s) \land t[1] = s[1] \land t[2] = \{u \mid u \in s[2] \land u[2] < 5\})$. In order to translate this formula into a relational algebra expression, we need to construct relational algebra expressions for $\text{ATOMS}(\psi)$, $\text{Dom}_t(\psi)$, and $\text{Dom}_u(\psi)$. $\text{ATOMS}(\psi) = \pi_1(Q) \cup \pi_2(Q) \cup \pi_3(Q)$ which equals $\{a,b,c,d,e,f,g,1,2,3,5,8,9\}$. The variable $u$ has scheme tuple: $(\text{atom}, \text{atom})$. $\text{Dom}_u(\psi)$ equals $\text{ATOMS}(\psi) \times \text{ATOMS}(\psi)$. The variables $s$ and $t$ have scheme tuple: $(\text{atom}, \text{relation} : (\text{atom}, \text{atom}))$ and hence, $\text{Dom}_t(\psi)$ equals $\text{Dom}_u(\psi)$. Both are equivalent to $\text{ATOMS}(\psi) \times P(\text{ATOMS}(\psi) \times \text{ATOMS}(\psi))$.

8. EQUIVALENCE OF CALCULUS AND ALGEBRA

There are two ways of computing new relations from old ones. One uses relational algebra expressions and the other uses safe calculus formulas having one free variable. The formulation of an equivalence theorem must be done with care. The calculus can express the full powerset with a single formula which is independent of the relation. It is impossible to do this in the algebra. However, given an interpretation of the base relations, there is a single algebraic expression which results in the powerset of the input relation. The expression for the powerset operator depends upon the cardinality of the evaluation of the constant relations and variable relations occurring in the expression representing the input relation. We prove the equivalence theorem in two steps: Theorem 8.1 translates algebra expressions into calculus formulas, and Theorem 8.2 translates from the calculus to the algebra.

THEOREM 8.1. Given any expression $E$ in the relational algebra, there is a safe formula $\psi_E$ in the calculus such that for any evaluation $\text{EV}$ of the relation variables and constant relations, the set of $t$ satisfying $\psi_E$ expresses the same relation. That is, $\{t \mid \psi_E(t)\} = \text{EV}(E)$.

PROOF. The proof is by induction on the number of operators in $E$.

Part I. Basis: $E$ Has No Operators:

Case a: $E$ expresses a constant relation. We use induction on the order of the scheme of $E$.

Order 1:

The scheme of $E$ is relation: $(\text{atom}, \ldots, \text{atom})$ with arity $n$. Suppose $E$ represents the set $\{t_1, \ldots, t_m\}$. Let $E_i$ represent the set consisting of the tuple $t_i$. Then $\text{EV}(E_i)$ equals $t_i$ where $\psi_i(t)$ is the formula $(t[1] = t_i[1] \land \cdots \land t[n] = t_i[n])$. It is easy to see that $\psi_i$ is a safe formula. Since $E$ is a union of the $E_i$ where $i$ varies from 1 to $m$, we can conclude that $E$ and the safe formula, $\psi_1(t) \lor \cdots \lor \psi_m(t)$, represent the same set.

Order $k+1$:

The scheme of $E$ is relation: $(e(1), \ldots, e(n))$. Each $e(i)$ is either an atom or a relation scheme of order $k$ or less. Suppose the set represented by $E$ consists of $m$ tuples $\{t_1, \ldots, t_m\}$ of arity $n$. Let $E_i$ be the expression representing $\{t_i\}$, then as before $E$ is equivalent to $E_1 \cup \cdots \cup E_m$. The components of each $t_i$ are either atoms or relations of order $k$ or less. By the induction hypothesis, any constant relation expressing a relation of order $k$ or less can be expressed by a safe calculus expression. Hence, given any $i$ between 1 and $m$, and any $j$ between 1 and $n$, if $e(j)$ is a relation scheme, then there is a safe formula $\psi_{ij}(t)$ such that $t_i[j]$ equals $\{s \mid \psi_{ij}(s)\}$. If $e(j)$ is not a relation, then the translation is even simpler, namely $t_i[j]$ equals $c_{i,j}$ where $c_{i,j}$, is an atomic...
constant. For each \( j \) for which \( e(j) \) is a relation, let \( \lambda_{i,j}(t) \) be the formula \( (t[j] = s \mid \psi_{i,j}(s)) \). If \( e(j) \) is an atom, let \( \lambda_{i,j}(t) \) be the formula \( (t[j] = c_{i,j}) \). Thus the constant relation represented by \( E_i \) is equivalent to \( \{ t \mid \psi_i(t) \} \), where \( \psi_i(t) \) is the formula \( \lambda_{i,1}(t[1]) \land \cdots \land \lambda_{i,n}(t[n]) \). It follows that \( E \) and \( \psi_1(t) \lor \cdots \lor \psi_m(t) \) express the same set. The safety of this formula is insured by the safety of its component parts, \( \lambda_{i,j}(t) \).

Case b: \( E \) is the relation variable \( P \). The proof of this is straightforward.

**Part II. Induction Step: \( k+1 \) Operators**

Assume \( E \) contains at least one operator, and the statement in the theorem is true for all expressions having \( k \) or fewer operators. We make a case by case analysis. The beginning cases mirror Ullman's proof [15].

Case a: \( E = E_1 \cup E_2 \), where \( E_1 \) and \( E_2 \) have \( k \) or fewer operators. According to the induction hypothesis, \( E_1 \) and \( E_2 \) can be equivalently represented by the safe formulas \( \psi_1 \) and \( \psi_2 \) with the same free variables, respectively. Therefore \( E \) can be represented by \( (\psi_1 \lor \psi_2) \) which is safe.

Case b: \( E = E_1 \cap E_2 \), where \( E_1 \) and \( E_2 \) have \( k \) or fewer operations. According to the induction hypothesis, \( E_1 \) and \( E_2 \) can be equivalently represented by the safe formulas \( \psi_1 \) and \( \psi_2 \), respectively. Therefore \( E \) can be represented by \( (\psi_1 \land \psi_2) \) which is safe.

Case c: \( E = E_1 - E_2 \), where \( E_1 \) and \( E_2 \) have \( k \) or fewer operations. According to the induction hypothesis, \( E_1 \) and \( E_2 \) can be equivalently represented by the safe formulas \( \psi_1 \) and \( \psi_2 \), respectively. Therefore \( E \) can be represented by \( (\psi_1 \land \neg \psi_2) \) which is safe.

Case d: \( E = E_1 \times E_2 \), where \( E_1 \) and \( E_2 \) have \( k \) or fewer operations. According to the induction hypothesis, \( E_1 \) and \( E_2 \) can be equivalently represented by the safe formulas \( \psi_1 \) and \( \psi_2 \), respectively. Suppose \( E_1 \) has scheme relation: \( (r(1), \ldots, r(n)) \) and \( E_2 \) has scheme relation: \( (s(1), \ldots, s(m)) \). Let \( \psi(t) \) be the following formula: \( \exists u \exists w (\psi_1(u) \land \psi_2(w) \land t[1] = u[1] \land \cdots \land t[n] = u[n] \land t[n+1] = w[1] \land \cdots \land t[n+m] = w[m]) \). Then \( E \) and \( \psi \) represent the same set.

Case e: \( E = \pi_Y(E_1) \), where \( E_1 \) has scheme relation: \( (r(1), \ldots, r(n)) \) and \( Y = \{ i_1, \ldots, i_k \} \) is a subset of the indexes \( \{1, \ldots, n\} \). In this case, \( E \) has scheme relation: \( (e(1), \ldots, e(k)) \) where \( e(s) \) is the same scheme as \( r(i_s) \) for \( 1 \leq s \leq k \). Using the induction hypothesis we have a formula, \( \psi_1(t) \), which represents the same set as \( E_1 \). Let \( \psi(t) \) be the formula \( \exists u (\psi_1(u) \land t[1] = u[i_1] \land \cdots \land t[k] = u[i_k]) \). Then \( E \) and \( \psi \) represent the same set.

Case f: \( E = \sigma_F(E_1) \), where \( E_1 \) has relation: \( (e(1), \ldots, e(n)) \) and \( F \) is a formula given in the definition of the selection operation in Section 7. By the induction hypothesis, there is a safe formula, \( \psi_1(t) \), representing the same set as \( E_1 \). Let \( \psi(t) \) be the formula \( \psi_1(t) \land F'(t) \) where \( F'(t) \) is \( F \) with each operand that denotes the component \( i \) replaced by \( t(i) \). Since \( F' \) contains no quantifiers, it is easy to check that \( F' \) is safe. All but one of the operations in the selection formula are also contained in the calculus language. The exception is \( \neq \). However, that comparison can expressed using \( = \) and logical negation.

Case g: \( E = \nu_Y(E_1) \), where \( E_1 \) has scheme relation: \( (r(1), \ldots, r(n)) \) and \( Y = \{ i_1, \ldots, i_k \} \) is a subset of the indexes \( \{1, \ldots, n\} \). Let \( X = \{ h_1, \ldots, h_{n-k} \} \) be the complement of \( Y \). Then \( E \) has scheme relation: \( (e(1), \ldots, e(n-k+1)) \), where \( e(i) = r(h_i) \) for each \( i \) between 1 and \( n-k \), and \( (n-k+1) \) has scheme relation: \( (t(1), \ldots, t(k)) \), with \( t(i) = r(i_j) \) for each \( j \) between 1 and \( k \). By the induction hypothesis, there is a safe formula, \( \psi_{i,j}(t) \), representing the same set as \( E_1 \). Let \( \psi \) be the formula defined by the following sequence of steps:

\[
\psi(t) = \exists u (\psi_2(t, u)),
\psi_2(t, u) = (\psi_1(u) \land t[1] = u[h_1] \land \cdots \land t[n-k] = u[h_{n-k}] \land t[n-k+1] = \{ w \mid (\psi_3(w, u)) \}),
\psi_3(w, u) = \exists r (\psi_4(w, r, u)),
\psi_4(w, r, u) = \psi_1(r) \land r[h_1] = u[h_1] \land \cdots \land r[h_{n-k}] = u[h_{n-k}] \land w[1] = r[i_1] \land \cdots \land w[k] = r[i_k].
\]

In words, \( \psi_2 \) says that \( u \) is a tuple in \( E_1 \), which agrees with \( t \) on the columns in \( X \) and whose last column is the set of values, \( w \), which together with \( u \) satisfy \( \psi_3 \). \( \psi_3(w, u) \) is satisfied for all \( k \)-tuples, \( w \), which occur as the \( Y \) columns of an \( n \)-tuple \( r \) in \( E_1 \) and whose \( X \) columns agree with \( u \).
The safety of ψ is verified by checking two conditions. The first condition, \{t \mid \psi(t)\} is a subset of Dom_ψ(ψ), holds because the atoms in t are the same as the atoms of ψ. The second condition states that existentially qualified subformulas should also be safe. ∃u(ψ_2(t, u)) is safe because \{u \mid ψ_2(t, u)\} is a subset of Dom_ψ(ψ_2) for any assignment of t. Furthermore, any u satisfying ψ_2 must also satisfy ψ_1, and by the safety of ψ_1, such a u is an element of Dom_ψ(ψ_1). Similarly, ∃r(ψ_4(w, r, u)) is also a safe subformula, since \{r \mid ψ_4(w, r, u)\} is a subset of Dom_ψ(ψ_4) for any assignment of w and r. Besides, any r satisfying ψ_4 must also satisfy ψ_1 and, by the safety of ψ_1, be an element of Dom_ψ(ψ_1).

Case h: E = μ_ψ(E_1) where E_1 has scheme relation: \( (t(1), \ldots, t(n)) \) and t(h) has scheme relation: \( (r(1), \ldots, r(m)) \). Let E have scheme relation: \( (e(1), \ldots, e(m + n - 1)) \) and let \( Y = \{i_1, \ldots, i_{n-1}\} \) denote the indexes from 1 to n with h omitted. Then \( e(i_j) = t(j) \) for any j between 1 and n - 1 and unequal to h. For k between n and m + n - 1, \( e(k) = r(k - n + 1) \). By the induction hypothesis, there is a safe formula, \( ψ_1(t) \), representing the same set as E_1. Let \( ψ \) be the formula:

\[ \exists r(ψ_1(r) \land t[1] = r[i_1] \land \cdots \land t[n-1] = r[i_{n-1}] \land \langle s \in r[h] \land t[n] = s[1] \land \cdots \land t[n+m-1] = s[m] \rangle) \]

This formula and E represent the same set. The proof that ψ is safe is similar to the argument given in Case g of this proof.

**Theorem 8.2.** Given any safe formula \( ψ(t) \) in the calculus and an interpretation of the predicate relations I, there is an expression \( E_ψ,I \) in the relational algebra such that \( \{t \mid ψ(t)\} = EV(E_ψ,I) \)

**Proof.** Given an interpretation of the predicate symbols in calculus (an instance of the database) and a formula \( ψ(t) \) there is an expression \( E_1(ψ) \) in the algebra such that \( E_1(ψ) \) represents \( D_1(ATOMS(ψ)) \). This follows directly from Lemma 7.8. Since \( ψ \) is fixed throughout the following proof, we abbreviate the set \( D_1(ψ) \) and the expression \( E_1(ψ) \) to \( D_1 \) and \( E_1 \), respectively, for any scheme s. We may assume that \( ψ \) only contains the operators: \( \lor, \land, \neg, \exists \) and the set formator \( r[i] = \{s \mid co(s, t, n, v, \ldots)\} \).

**Part I. Basis: Zero Operators:**

Suppose \( ω \) is a subformula of ψ containing no logical operators and no quantifiers. In the following, assume that s has scheme tuple: \( (s(1), \ldots, s(n)) \) and t has scheme tuple: \( (t(1), \ldots, t(m)) \).

Case a. \( ω \) has a predicate symbol \( P \). Then \( ω(s) \) must be \( P(s) \) and \( D_1 x D_t \) \( \{\langle s, t \rangle \mid ω(s, t)\} \) is expressible by \( E_1 \).

Case b. \( ω \) uses a comparison operator. We have two subcases.

Subcase (i). \( ω(s, t) \) is \( s[i] \circ \circ t[j] \), where \( \circ \circ \) is one of \( =, \leq, \geq, >, < \), and schemes \( s(i) \) and \( t(j) \) are compatible with \( \circ \circ \). Then \( D_s x D_t \) \( \{\langle s, t \rangle \mid ω(s, t)\} \) is expressed by \( σ_F(E_s x E_t) \), where \( F \) is the formula \( i \circ \circ j \).

Subcase (ii). \( ω(s) \) is \( s[i] \circ \circ c \), where \( \circ \circ \) is as above and \( c \) is a constant with scheme \( s(i) \). Then \( D_s \) \( \{\langle s \rangle \mid ω(s)\} \) is expressed by \( σ_F(E_s) \), where \( F \) is the formula \( i \circ \circ c' \).

Case c. \( ω \) uses the membership test. There are two subcases.

Subcase (i). \( ω(s, t) \) is \( s \in t[j] \), where \( t[j] \) is an indexed variable with scheme relation: \( s \). Then \( D_s x D_t \) \( \{\langle s, t \rangle \mid ω(s, t)\} \) has arity \( n + m \) and is expressed by \( (σ_F(E_s x E_t)) \) where \( F \) is the formula \( 1, \ldots, n \) \( \in j \). This derived selection operation was proven in Lemma 7.3.

Subcase (ii). \( ω(s, t) \) is \( s[i] \in t[j] \) where the schemes are as above. In this case, \( D_s x D_t \) \( \{\langle s, t \rangle \mid ω(s, t)\} \) is expressed by \( (σ_F(E_s x E_t)) \) where \( F \) is the formula \( i \in n + j \). This is another derived selection operator given in Lemma 7.3. In Subcases (i) and (ii), we can replace either \( s \) or \( t[j] \) by a constant and use the same method to translate formulas involving constants.

**Part II. The Induction Step: k+1 Operators.**

The induction hypothesis states that if \( ω_1(s_1, \ldots, s_n) \) is a subformula of \( ψ(t) \) with k or fewer operators, then \( D_{s_1} \times \cdots \times D_{s_n} \) \( \{\langle s_1, \ldots, s_n \rangle \mid ω_1(s_1, \ldots, s_n)\} \) is expressible in the relational algebra. We must show that the same is true for any subformula with \( k + 1 \) logical operators.
Case a. \( \omega(y_1, \ldots, y_n) \) is \( \omega_1(s_1, \ldots, s_h) \lor \omega_2(x_1, \ldots, x_v) \) where \( \omega_1 \) and \( \omega_2 \) each have \( k \) or fewer operators. Set \( Y = \{y_1, \ldots, y_n\} \) equal to the free variables of \( \omega \), \( S = \{s_1, \ldots, s_h\} \) equal to the free variables of \( \omega_1 \), and \( X = \{x_1, \ldots, x_v\} \) equal to the free variables of \( \omega_2 \). \( Y \) equals the union of \( S \) and \( X \), but \( S \) and \( X \) need not be disjoint. By the induction hypothesis there are algebraic expressions \( E_1 \) and \( E_2 \) representing

\[
\begin{align*}
D_{y_1} \times \cdots \times D_{y_n} \cap \{(s_1, \ldots, s_h) \mid \omega_1(s_1, \ldots, s_h)\}, \quad \text{and} \\
D_{x_1} \times \cdots \times D_{x_v} \cap \{(x_1, \ldots, x_v) \mid \omega_2(x_1, \ldots, x_v)\},
\end{align*}
\]

respectively.

Then, \( D_{y_1} \times \cdots \times D_{y_n} \cap \{(y_1, \ldots, y_n) \mid \omega(y_1, \ldots, y_n)\} \) is expressible as the union of two expressions \( W_1 \) and \( W_2 \), where \( W_1 \) is \( E_1 \) with a column \( D_{y_i} \) added for each variable \( y \) occurring in \( Y \) but not in \( S \), and likewise, \( W_2 \) is \( E_2 \) with a column \( D_{x_j} \) added for each variable \( x \) occurring in \( X \) but not in \( X \). This construction can be found in Ullman’s proof [15].

Case b. \( \omega(y_1, \ldots, y_n) \) is \( \omega_1(s_1, \ldots, s_h) \land \omega_2(x_1, \ldots, x_v) \). Conjunction is handled exactly as disjunction except that the intersection of the two expressions \( W_1 \) and \( W_2 \) is formed, instead of their union.

Case c. \( \omega(s_1, \ldots, s_n) \) is \( \neg \omega_1(s_1, \ldots, s_n) \), where \( \omega_1 \) has \( k \) operators. By the induction hypothesis, there is an algebraic expression \( E_1 \) representing \( D_{s_1} \times \cdots \times D_{s_n} \cap \{(s_1, \ldots, s_n) \mid \neg \omega_1(s_1, \ldots, s_n)\} \). Then, \( D_{s_1} \times \cdots \times D_{s_n} - E_1 \) is an algebraic expression for \( \{(s_1, \ldots, s_n) \mid \omega(s_1, \ldots, s_n)\} \). Case d. \( \omega(s_1, \ldots, s_n) \) is \( \exists_{n+1}(\omega_1(s_1, \ldots, s_{n+1})) \) where \( \omega_1 \) has \( k \) or fewer operators. Suppose scheme \( s_t \) has \( c_t \) columns. Then \( D_{s_1} \times \cdots \times D_{s_{n+1}} \) has arity \( c_1 + \cdots + c_{n+1} \) and \( D_{s_1} \times \cdots \times D_{s_n} \) has arity \( N = c_1 + \cdots + c_n \). By the induction hypothesis, there is an algebraic expression \( E_1 \) for \( D_{s_1} \times \cdots \times D_{s_{n+1}} \cap \{(s_1, \ldots, s_{n+1}) \mid \omega_1(s_1, \ldots, s_{n+1})\} \). Since \( \omega_1 \) occurs in an existential subformula of the original safe formula \( \omega \), \( \omega_1(s_1, \ldots, s_{n+1}) \) can only be true if \( s_{n+1} \) is in \( D_{s_{n+1}} \). Thus, we can conclude that \( \pi_{\{1, \ldots, N\}}(E_1) \) is an algebraic expression representing \( \{(s_1, \ldots, s_n) \mid \omega(s_1, \ldots, s_n)\} \).

Case e. \( \omega(s_1, \ldots, s_n) \) is \( \forall_{n+1}(\omega_1(s_1, \ldots, s_{n+1})) \) where \( \omega_1 \) has \( k \) operators. Suppose scheme \( s_t \) has \( c_t \) columns. Then \( D_{s_1} \times \cdots \times D_{s_{n+1}} \) has arity \( c_1 + \cdots + c_{n+1} \) and \( D_{s_1} \times \cdots \times D_{s_n} \) has arity \( N = c_1 + \cdots + c_n \). By the induction hypothesis, there is an algebraic expression \( E_1 \) representing \( D_{s_1} \times \cdots \times D_{s_{n+1}} \cap \{(s_1, \ldots, s_{n+1}) \mid \omega_1(s_1, \ldots, s_{n+1})\} \). Let \( Z \) be the last set by \( W \). If the scheme for the tuple variable \( t_i \) has \( c_t \) columns, then \( W \) has arity \( n + c_1 + \cdots + c_n \). Let \( Z \) be \( D_{s_1} \times D_{s_1} \times \cdots \times D_{s_n} \cap \{(s, t_1, \ldots, t_n) \mid \omega(s, t_1, \ldots, t_n)\} \), which also equals \( D_{s_1} \times D_{s_1} \times \cdots \times D_{s_n} \cap \{(s, t_1, \ldots, t_n) \mid s[i] = s[i] \} \). \( Z \) has arity \( d + c_1 + \cdots + c_n \). \( Z \) can be represented by the following sequence of algebraic operations:

1. Nest \( W \) on the \( u \) columns, which in fact are the first \( m \) columns,
   \( W_1 = \nu_{1, \ldots, m}(W) \). \( W_1 \) has arity \( 1 + c_1 + \cdots + c_n \).
2. Add an \( s \) column using Cartesian product,
   \( W_2 = E_s \times W_1 \). \( W_2 \) has arity \( d + 1 + c_1 + \cdots + c_n \).
3. Select those tuples in \( W_2 \) whose \( i \)th column equals the nested \( u \)th column,
   \( W_3 = \sigma_{F}(W_2) \), where \( F \) is the formula \( i = d + 1 \).
4. We must get rid of the duplicated \( u \)th and \( i \)th columns, \( Z = \pi_Y(W_3) \), where \( Y \) are all the columns in \( W_3 \) from 1 to \( d + 1 + c_1 + \cdots + c_n \) excluding the \( (d + 1)^{st} \) column which now duplicates the \( i \)th column. That is, \( Y = \{1, 2, \ldots, d, d + 2, \ldots, d + 1 + c_1 + \cdots + c_n \} \). This produces the desired set, with the correct arity.

This completes the proof of Theorem 8.2.

9. AN EXAMPLE

In this section, we trace the proof of the equivalence of algebra and calculus. Given the instance in Figure 2 of scheme \( Q \), consider the query, "Eliminate from \( Q \) those \( R \) columns whose second component has a value greater than or equal to five."

9.1. Calculus to Algebra Translation

A calculus expression for this query:

\[
\psi(t) = \exists_s(Q(s) \land t[1] = s[1] \land t[2] = \{u \mid u \in s[2] \land u[2] < '5'\}).
\]
Example 7.5 gives the algebraic expressions $E_u, E_s, E_t$ for the domains $D_u, D_s, D_t$. Tracing Theorem 8.2, we first translate subformulas of $\psi$ having no logical operators, then all subformulas of $\psi$ with one logical operator, and so on. Finally, the unique subformula with 5 logical operators, which is just $\psi$ itself, is translated.

Step 1: Subformulas with no logical operators:

a. $u \in s[2]$ is translated by $\sigma_{1 \in s}(E_u \times E_t)$ (Part I, Case c).

b. $u[2] \lessdot '5'$ is translated by $\sigma_{2 < '5'}(E_u)$ (Part I, Case b).

c. $t[1] = s[1]$ is translated by $\sigma_{1 = 3}(E_t \times E_s)$ (Part I Case b).

d. $Q(s)$ is translated by $Q$ (Part I, Case a).

Step 2: Subformulas with one logical operator:

a. $u \in s[2] \land u[2] \lessdot '5'$ is translated by $E_1 = \sigma_{1 \in s}(E_u \times E_t) \cap (\sigma_{2 < '5'}(E_u) \times E_s)$ (Part II, Case d).

b. $Q(s) \land t[1] = s[1]$ is translated by $E_2 = E_t \times (Q_{1 = 3}(E_t \times E_s)$ (Part II, Case d).

Step 3: There is one subformula with two logical operators.

t[2] = \{ u \in s[2] \land u[2] \lessdot '5' \}$ is translated by $E_3 = \pi_{\{1,3,4,5\}}(\sigma_{2 = 3}(E_t \times \nu_{\{1,2\}}(E_1)))$ (Part II, Case e).

In tracing this example, it is important to keep track of the arity of the schemes. The relation expressed by $E_1$ has arity 4 since both $E_u$ and $E_t$ have arity 2. Hence, the first step $\nu_{\{1,2\}}(E_1)$ nests $E_1$ on its first two columns which are in effect nesting on column $u$ resulting in a scheme with arity 3. The next step adds a $t$ column to create the relation $E_t \times \nu_{\{1,2\}}(E_1)$ which has arity 5. Selection on $2 = 3$, selects those tuples whose $t[2]$ component equals the nested $u$ column. The last step is a projection to get rid of column 2 which is now a duplicate of column 3.

Step 4: There is one subformulas with 4 logical operators. (There are no subformulas with 3 operators.) $(Q(s) \land t[1] = s[1] \land t[2] = \{ u \in s[2] \land u[2] \lessdot '5' \})$ is translated by $E_2 \cap E_3$. Both $E_2$ and $E_3$ have schemes with arity 4. Thus, the resultant scheme has arity 4.

Step 5: There is one subformulas with 5 operators and that is $\psi$ itself. $\exists s(Q(s) \land t[1] = s[1] \land t[2] = \{ u \in s[2] \land u[2] \lessdot '5' \})$ is translated by $\pi_{\{1,2\}}(E_2 \cap E_3)$ (Part II, Case d).

This completes the trace of Theorem 8.2. This is not an efficient algebraic expression, but it works. A more efficient algebraic expression for expressing this query is given in Example 7.5.

9.2. Algebra to Calculus Translation

Tracing Theorem 8.1, we translate the following algebra expression for the same query

$\pi_{\{3,4\}} \nu_{\{4,5\}} \sigma_{5 < '5'} \mu_4(Diag(Q))$.

Step 1. Diag($Q$) is shorthand for $\sigma_F(Q \times Q)$, where $F$ is the formula $(1 = 3 \land 2 = 4)$. $Q \times Q$ is translated to $\psi_0(t)$ where $\psi_0(t)$ is the formula $\exists u(\exists v(Q(u) \land Q(v) \land t[1] = u[1] \land t[2] = u[2] \land t[3] = v[1] \land t[4] = v[2])$ (Part II, Cased). Diag($Q$) is translated to $\psi_1(t)$, where $\psi_1(t)$ is the formula $\psi_0(t) \land t[1] = t[3] \land t[2] = t[4]$ (Part II, Case f).

Step 2. $\mu_4(Diag(Q)$ is translated by $\psi_2(t)$, where $\psi_2(t)$ is the formula $\exists r \exists s(\psi_1(r) \land t[1] = r[1] \land t[2] = r[2] \land t[3] = r[3] \land (s \in r[4] \land t[4] = s[1] \land t[5] = s[2]))$ (Part II, Case h). Unnesting on the fourth column of Diag($Q$) increased the arity by 1 since the fourth column was a relation containing 2-tuples. Hence $t$ is a 5-tuple.

Step 3: $\sigma_{5 < '5'} \mu_4(Diag(Q)$ is translated by $\psi_3(t)$, where $\psi_3(t)$ is the formula $\psi_2(t) \land t[5] < '5'$ (Part II, Case f).


Step 5: $\pi_{\{3,4\}} \nu_{\{4,5\}} \sigma_{5 < '5'} \mu_4(Diag(Q)$ is translated by $\psi(t)$, where $\psi(t)$ is the formula $\exists u(\psi_4(u) \land t[1] = u[3] \land t[2] = u[4])$ (Part II, Case d).

This finishes the translation from algebra to calculus.
10. GENERALIZING THE EQUIVALENCE PROOF

The relational calculus with the membership and set formator formulas can express the powerset of a relation. On the other hand, the relational algebra with nest and unnest operators can not express the powerset of a relation unless an instance for that relation is provided. Depending on the relation instance, a different algebra expression is generated for the powerset. So, this extended relational calculus is more powerful than the relational algebra with nest and unnest operators. We have shown that the relational calculus and algebra as defined in this work have the same expressive power provided an instance of the database is specified. This is the strongest possible equivalence assertion which can be proven. That is, any relational algebra expression can be translated to a relational calculus formula. But, a relational calculus formula cannot, in general, be translated to relational algebra unless a database instance is given.

We now show how to augment the relational algebra language so that it becomes as expressive as the relational calculus language for nested relations. There are several alternatives. One possibility consists of simply adding the power set operator to the relational algebra. A different approach consists of adding a while loop [19], an assignment operator and a comparison operator to the relational algebra with the nest and unnest operations. This algebra has the same expressive power as the relational calculus for nested relations. The following is an algorithm which creates the full powerset for an algebra expression $E$.

Algorithm 1

\[
\begin{align*}
\text{k} &:= 1 \\
\text{While } P_k(E) \neq P_{k+1}(E) \text{ do} \\
\quad \text{k} &:= k + 1 \\
\text{Endwhile}
\end{align*}
\]

LEMMA 10.1. Algorithm 1 calculates the powerset of the relational algebra expression $E$.

PROOF. Algorithm 1 calculates $P_1(E), P_2(E), \ldots$. Eventually it calculates $P_n(E)$ where $n$ is the cardinality of the relation created by $E$. Note that $P_n(E)$ is the $n^{th}$ restricted power set operator which computes $n$ or less subsets of $EV(R)$. Algorithm 1 always halts. Since the relation instances in the database are finite, the while loop in Algorithm 1 is executed a finite number of times. For any $k, k \leq n, P_k(E) \neq P_{k+1}(E)$ because $P_k(E)$ contains subsets of $E$ with $k$ or fewer elements, whereas $P_{k+1}(E)$ has subsets with $k + 1$ or fewer elements. On the other hand, when $k$ reaches $n$, $P_k(E)$ and $P_{k+1}(E)$ contain the same subsets because the $n$ or fewer and $n + 1$ or fewer subsets of $EV(E)$ are the same, since $E$ only contains a total of $n$ tuples.

THEOREM 10.1. Given any safe formula $\psi(t)$ in the relational calculus, there is an equivalent expression $E$ in the relational algebra, i.e., $\{t \mid \psi(t)\} = EV(E)$.

PROOF. $\text{Dom}_k(\psi(t))$ is expressible in the relational algebra. Use Algorithm 1 instead of the restricted power set operator $P_k$ in Lemmas 7.7 and 7.8. The $\text{Dom}_k(\psi(t))$ will be generated independently of the interpretations of the symbols in $\psi(t)$. The rest is the same as the proof of Theorem 8.2.

Another alternative is adding a power set operator to the relational algebra. Such a relational algebra has the same expressive power as the relational calculus. The power set operator is used in Lemmas 7.7 and 7.8 to form $\text{Dom}_k(\psi(t))$. So, the translation from the relational calculus to the relational algebra will not be instance dependent. Theorem 6.1 gives the relational calculus expression for the power set operation of the relational algebra. This can be added as another case to the proof of Theorem 8.1. There is a subtle difference between the former and latter approaches. Algorithm 1 generates a different expression for each instance of $E$. However, the work is done within the algorithm and it is transparent in the calculus to algebra translation. On the other hand, adding the power set operator creates a fixed expression in the translation from calculus to algebra, which is independent of any particular instance. In either approach, translation from the algebra to the calculus always results in an expression which is instance independent.
11. CONCLUSION

In this paper, we define algebra and calculus languages for nested relations. A formal inductive definition is given, which is a generalization and refinement of previous definitions in the literature [3,7,9]. It is based on the tuple and relation distinction, and quantitatively assigns an order to a nested structure. The algebra and calculus languages we give are quite similar to the ones in the above mentioned studies.

We first show a limited equivalence between these languages since the translation (proof) from the calculus to the algebra requires an interpretation of the relations mentioned in the query expression. This is the strongest equivalence proof possible for relational algebra and calculus for nested relations. We later add more power to the relational algebra by augmenting it with programming constructs and provide an algorithm to construct the full power set of a relation independent of its instance. Then, we show that this augmented relational algebra has the same expressive power as the relational calculus; this equivalence does not depend upon the interpretations of the symbols mentioned in the algebra and calculus expressions.

REFERENCES