Relations with Relation Names as Arguments: Algebra and Calculus

Kenneth A. Ross
Columbia University
kar@cs.columbia.edu

Abstract

We consider a version of the relational model in which relation names may appear as arguments of other relations. Allowing relation names as arguments provides enhanced modelling capabilities, allowing some object-oriented features to be expressed within the relational model. We extend relational algebra with operators for accessing relations, and also define a relational calculus based on the logic HiLog. We prove two equivalence results between extensions of relational algebra and fragments of our HiLog-based calculus. We show that our extensions of relational algebra provide higher expressive power than relational algebra on any given database. Finally, we argue that the extensions proposed here are relatively easy to provide in practice, and should be expressible within modern query languages.

1 Introduction

Since the relational approach was proposed in the early 1970's [Cod70] it has become the defacto standard as a basis for designing real-world database systems. Data are stored in relations that may be queried using either relational algebra or a relational calculus. Query languages such as SQL [C$^+$76] are based on relational calculus, with queries being translated (with optimization) into operations on the relations in relational algebra.

One of the fundamental results of relational database theory is that exactly the same class of queries can be expressed by relational algebra and the "safe" formulas of relational calculus [Cod72].[1] A translation from queries in one formalism to equivalent queries in the other can be constructed. It was suggested by Codd that any relational query language should have at least the expressive power of these formalisms [Cod72].

More recently, researchers have been considering the object-oriented database paradigm, in which features such as structured objects, classes and inheritance are available. Such features can be very naturally expressed if we allow *relation names* to be arguments of other relations, and allow access to one relation or another based on the value of some other attribute. For example, sets can be represented by creating a unary relation that holds for all members of the set, and substituting the relation name as an argument where required. In this way nested relations can exist without having relation *extensions* in tuples.

The idea of having relation names as arguments goes back to [SS77], where the authors demonstrate how class hierarchies can be naturally modelled using this principle. In addition the authors examine issues of referential integrity. They briefly describe an algebraic operator called `SPECIFY` that converts a single relation name into its relation, but they do not consider the more general problem of converting a set of relation names into relations.

First order logic is not an appropriate basis for a calculus on relations that have relation names as arguments. In first order logic, predicate symbols and constant symbols come from disjoint domains, and cannot be mixed. Recently, a logic that does allow this mixing has been proposed. This logic is called HiLog [CKW89]. One may think of HiLog as a version of first order logic in which there is no distinction between variables, function, constant and predicate symbols. In particular, variables may appear in predicate names.

In this paper we propose two versions of re-

[1] See Section 3.1 for a definition of safety.

lational algebra for relations that allow relation names as arguments. The first, which we call "E-relational algebra," extends standard relational algebra with a set of *expansion* operators. These operators expand a set of relation names into the the union of their relations (with the relation name as an extra argument). The second, which we call "T-relational algebra," extends E-relational algebra by with a set of *totality* operators. These operators allow access to the names of all nonempty relations (of a given arity) that are present in the database.

We also propose an extension of the relational calculus using HiLog as a basis rather than first order logic; we call the extension "relational HiLog." We generalize the notion of safety in two ways to relational HiLog, obtaining notions of "weak safety" and "strong safety."

The main technical results of this paper are

- The equivalence of E-relational algebra and the class of strongly safe relational HiLog expressions.
- The equivalence of T-relational algebra and the class of weakly safe relational HiLog expressions.

We further show that for a given database, the set of queries that can be expressed by T-relational algebra expressions is strictly greater that those expressible by E-relational algebra expressions, which is in turn strictly greater than those expressions that can be constructed using standard relational calculus.

We briefly discuss the issues of optimizing our extended algebras, how schema evolution may be affected, and how relational HiLog may be extended to a deductive database language.

The modelling capability of relations with names as arguments significantly extends that of the standard relational approach, as argued in [SS77]. We propose extensions of relational algebra and relational calculus that would not be difficult to implement *even within the framework of current relational database systems.* Structurally, relations remain first-order. We argue that relational systems could provide additional modelling capability simply by augmenting their calculus and algebra as described in this paper.

2 Algebra

We use the term *relational algebra* to denote the standard notion of relational algebra as introduced by Codd [Cod70]. The fundamental notions are union ($R \cup S$), difference ($R - S$), cross-product ($R \times S$), projection ($\pi_{i_1,\cdots,i_m}(R)$), and selection ($\sigma_F(R)$). Other operators, such as join and intersection, can be defined in terms of these.

We shall consider two additional classes of operator that we define below. Our extension of the standard relational algebra allows relation names to appear as arguments of other relations, and for reference to relations to be determined by the value of other relational expressions rather than having to be "hard-wired" into a relational expression.

In our model, every symbol appearing in the database can serve as a relation symbol. In fact, for every such symbol we shall assume that there is a relation of every arity greater than or equal to 1. While there are infinitely many such relations, we shall assume that there are only finitely many *nonempty* relations in the database.

The first operator we call an *expansion* operator, and is denoted by α^j, where $j \geq 1$.[2] An expansion operator (say α^k) expands a set of relation *names* into the union of the arity-k relation *extensions.*

Definition 2.1: Let R be a relational expression with one attribute. Then $\alpha^j(R)$ denotes the relation of arity $j+1$ defined by

$$\bigcup_{r \in R}(\{r\} \times r(X_1, \cdots, X_j))$$

It may sometimes be convenient to apply such an operator to an attribute of a relational expression that has more than one attribute. Let S be an arbitrary relational expression. We define

$$\beta_i^j(S) = \alpha^j(\pi_i(S))$$

Clearly, $\alpha^j(R) = \beta_1^j(R)$. □

Example 2.1: Let the relations p, q and r be as follows:

$$p: \begin{array}{c} p \\ q \\ r \end{array} \qquad q: \begin{array}{cc} a & b \\ b & c \\ d & d \end{array} \qquad r: \begin{array}{cc} b & c \\ d & p \end{array} \qquad r: \begin{array}{c} c \\ a \end{array}$$

and suppose that these are the only nonempty relations in the database. Note that r has extensions of more than one arity. Then $\alpha^2(p)$ is the relation

$$\begin{array}{ccc} q & a & b \\ q & b & c \\ q & d & d \\ r & b & c \\ r & d & p \end{array}$$

Note that the *binary* relation p is empty even though the *unary* relation p is not, and so there

[2] In fact, the collection $\{\alpha^1, \alpha^2, \cdots\}$ denotes an infinite class of expansion operators.

are no tuples with first argument p in $\alpha^2(p)$. Also, we look at the binary extension of r and not its unary extension. The value of $\beta_2^1(r)$ is

$$\begin{array}{cc} p & p \\ p & q \\ p & r \end{array}$$

Since the unary relation c is empty, the only tuples correspond to tuples from p. $\square$

A second type of operator we consider is the *totality* operator. The totality operator takes no arguments, and hence can be considered a special type of unary relation. For each $k \geq 1$ there is a totality operator ρ_k that holds for all relation names whose extensions at arity k are nonempty.

Using Example 2.1, $\rho_2 = \{q, r\}$, $\rho_1 = \{p, r\}$, and $\rho_i = \emptyset$ for $i \geq 3$. (It would be possible to adopt the convention that the nonempty ρ_i relations are also members of ρ_1. While we do not adopt this convention here, the results in this paper would not be affected if we were to do so.)

We shall consider two versions of relational algebra. The first is standard relational algebra together with the extension operators, which we call the *E-relational algebra*. The second, which we call the *T-relational algebra*, consists of the E-relational algebra together with the totality operators.

Note that relations themselves are structurally no different from before. The new features are the expansion and totality operators. The expansion operators are easy to apply; the important step is the translation from the relation name to the relation itself. As for the totality operators, it is reasonable to assume that the system will keep track of the names of the nonempty relations in the database. However, queries that refer to the totality operators may do an unrestricted lookup of the entire database (of relations having a particular arity). Clearly such lookups should be avoided when possible, but queries like "Tell me all unary relations that are true of a" necessarily have to do unrestricted lookups.

3 Calculus

In order to allow relation names to be available as arguments, we shall have to use a logic more general than relational calculus. Indeed, our logic should allow variables to appear as relation names. Such a logic, named HiLog, has recently been proposed [CKW89]. HiLog is more general than we require, since it allows uninterpreted function symbols. Hence we shall deal with a fragment of HiLog which we call *relational HiLog*. Relational HiLog is closely related to "DataHiLog," defined implicitly in [Ros91].

Definition 3.1: A relational HiLog *term* is either a variable or a constant symbol. If $t, t_1, \cdots, t_n$ are relational HiLog terms, where $n \geq 1$, then $t(t_1, \cdots, t_n)$ is a relational HiLog *atom*.[3] Relational HiLog *formulas* can be built from relational HiLog atoms in the usual way using the connectives $\vee$, $\wedge$, $\neg$, the quantifiers $\forall$, $\exists$, and parentheses. $\square$

Note that logical symbols are arityless. Unlike in full HiLog, there is a distinction between the class of terms and the class of atoms.

Example 3.1: The formula

$$p(X) \wedge \exists Y : X(Y)$$

is a relational HiLog formula that is true of X when X is a member of the relation p, and X is nonempty as a unary relation. The relational HiLog formula

$$\forall Y \forall Z : (\neg X(Y, Z) \vee X(Z, Y))$$

is true of X if X is a symmetric binary relation, i.e., if $X(a, b)$ holds then so does $X(b, a)$. $\square$

We can define a calculus based on relational HiLog in an analogous way to how domain relational calculus is defined in terms of first order function-free predicate calculus. We shall call this calculus the "relational HiLog calculus," or when the context is clear, simply "relational HiLog." We allow comparison operators, namely $=$, $<$, $>$, $\leq$, $\geq$ and $\neq$ as "built-in" binary relations.

Definition 3.2: A *relational HiLog expression* is of the form

$$\{x_1, \cdots, x_k | \psi(x_1, \cdots, x_k)\}$$

where ψ is a relational HiLog formula whose only free variables are the (distinct) variables $x_1, \cdots, x_k$. $\square$

3.1 Safety

We now consider the issue of safety. Recall that for domain relational calculus formulas ψ, DOM(ψ) denotes the unary relation consisting of all symbols appearing explicitly in ψ, together with all symbols appearing in each relation mentioned in ψ.

Definition 3.3: Let ψ be a domain relational calculus formula. A domain relational calculus expression $\{x_1, \cdots, x_k | \psi(x_1, \cdots, x_k)\}$ is defined to be *safe* [Ull89] if

[3] Since relations must have at least one attribute, we do not allow 0-ary terms.

1. $\psi(x_1, \cdots, x_k)$ true implies x_i is in DOM(ψ),

2. if $\exists u\ (\omega(u))$ is a subformula of ψ, then $\omega(u)$ true for any values of the free variables of ω (besides u) implies that u is in DOM(ω), and

3. if $\forall u\ (\omega(u))$ is a subformula of ψ, then $\omega(u)$ false for any values of the free variables of ω (besides u) implies that u is in DOM(ω).

□

Codd showed that relational algebra and the safe formulas of (tuple) relational calculus could express exactly the same class of queries [Cod72]. We shall do the same for two different extensions of relational algebra using the operators above. The extensions will be shown equivalent to appropriate subsets of relational HiLog.

What is the appropriate generalization of the notion of safety from that of domain relational calculus? One initial observation is that it is insufficient to consider only relations explicitly mentioned in the relational HiLog formula ψ when constructing DOM(ψ). Relation names may be variables, and so may not appear anywhere in ψ.

The following notions will be useful in the definitions below.

Definition 3.4: We say the *depth* of a relational HiLog formula ψ, denoted by $d(\psi)$ is the number of atoms in ψ with variables as names. Intuitively, $d(\psi)$ provides an upper bound on the number of times we have to recursively consider arguments of one relation as potential names of another. We define the *arity-set* of ψ, denoted $ar(\psi)$, to be the set of arities of atoms mentioned in ψ. The arity-set will always be finite, and will allow us to consider only a finite number of relations.[4] □

Definition 3.5: Let ψ be a relational HiLog formula. Let $\mathrm{DOM}_W(\psi)$ denote the unary relation consisting of all symbols appearing explicitly in ψ, together with all symbols appearing in each nonempty relation of arity in $ar(\psi)$ mentioned in the database (including the relation *names*). □

$\mathrm{DOM}_W(\psi)$ does not simplify to DOM(ψ) when ψ happens to be a formula of domain relational calculus, since DOM(ψ) is restricted to those relations mentioned in ψ. However, this distinction is not critical. If we were to redefine DOM(ψ) so that all relations (of appropriate arity) in the database were considered then the resulting notion of safety for domain relational calculus formulas (using Definition 3.3) would be the same as before.

An alternative definition of the domain is given in Definition 3.6 below.

Definition 3.6: Let ψ be a relational HiLog formula. Let $\mathrm{DOM}_S^0(\psi)$ denote the unary relation consisting of all symbols appearing explicitly in ψ, including all relation names mentioned in ψ. Let $\mathrm{DOM}_S^{i+1}(\psi)$ denote the unary relation consisting of all symbols appearing in relations of arity belonging to $ar(\psi)$ whose names belong to $\mathrm{DOM}_S^i(\psi)$, for $i = 0, \cdots, d(\psi)$. Define

$$\mathrm{DOM}_S(\psi) = \bigcup_{0 \le i \le d(\psi)+1} \mathrm{DOM}_S^i(\psi).$$

□

Informally, $\mathrm{DOM}_S(\psi)$ denotes the domain "reachable" from ψ using $d(\psi)$ expansions of arguments into relation names. $\mathrm{DOM}_S(\psi)$ could perhaps be made smaller by only expanding arguments using the arity with which they actually appear. However, the aspect of $\mathrm{DOM}_S(\psi)$ that is critical for our results is not its size, but that it can be (finitely) represented using a certain subset of relational HiLog.

Based on these two definitions of the domain, we get two different notions of safety. We simply apply Definition 3.3 to relational HiLog expressions with our new definitions of the domain. We shall call the version based on $\mathrm{DOM}_W(\psi)$ "weak safety," and the version based on $\mathrm{DOM}_S(\psi)$ "strong safety." (Note that they yield the same concept of safety for domain relational calculus formulas ψ.)

Example 3.2: The relational HiLog expression

$$\{X | \forall Y \forall Z : (\neg X(Y, Z) \vee X(Z, Y))\}$$

which returns all relation names that are symmetric as binary relations, is not even weakly safe, since all empty relations are symmetric, and there are potentially infinitely many empty binary relations. The relational HiLog expression

$$\{X | \exists Y : X(a, Y)\}$$

returns all relation names that, as binary relations, have a as the first argument of some tuple. This expression is weakly safe, but not strongly safe since X is not "reachable" from some fixed relation name mentioned in the formula. The relational HiLog expression

$$\{X | \exists Y : p(X) \wedge X(Y)\}$$

returns those elements of the p relation that are themselves nonempty as unary relations. This expression is strongly safe since the variable X is bound by the fixed relation p in the above formula. □

[4] Since every symbol has infinitely many possible arities, we would have to look for domain elements in infinitely many relations if we didn't restrict ourselves to a certain finite set of arities.

4 Equivalence of Algebra and Calculus

The following two lemmas show how both E-relational algebra and T-relational algebra can be reduced to relational HiLog.

Lemma 4.1: Every E-relational algebra expression has an equivalent expression in strongly safe relational HiLog.
Proof: (Sketch) The proof uses an inductive argument similar to that presented in [Ull89]. The induction is on the number of operators in the E-relational algebra expression. The operators union, cross-product, difference, projection and selection are handled as in [Ull89]. We show how to express $\alpha^j(R)$ in relational HiLog. By the induction hypothesis, there is some strongly safe relational HiLog expression $\{X|\psi(X)\}$ for R. Then $\alpha^j(R)$ will be represented by the relational HiLog expression

$$\{X, X_1, \cdots, X_j | \psi(X) \wedge X(X_1, \cdots, X_j)\}.$$

This expression can be shown strongly safe using the fact that the depth of this expression is one more than the depth of $\psi(X)$. ∎

Lemma 4.2: Every T-relational algebra expression has an equivalent expression in weakly safe relational HiLog.
Proof: (Sketch) The proof is similar to that of Lemma 4.1 except that our induction hypothesis is weaker, and we need to show how the totality operators can be expressed. Since the totality operators take no arguments, they can be handled in the base case. The operator ρ_j is expressed as the relational HiLog formula

$$\{X | \exists Y_1 \cdots \exists Y_j : X(Y_1, \cdots, Y_j)\}.$$

This formula is weakly safe because any X or Y_i that makes the formula above true must be a member of $\mathrm{DOM}_W(\psi)$.[5] ∎

We now examine the converse relationships, namely the simulation of strongly and weakly safe relational HiLog expressions respectively by E-relational algebra and T-relational algebra.

Lemma 4.3: Every strongly safe relational HiLog expression has an equivalent expression in E-relational algebra.
Proof: (Sketch) First we observe that $\mathrm{DOM}_S^0(\psi)$ can be expressed in T-relational algebra as a finite set of values. If R_i is an E-relational algebra expression for $\mathrm{DOM}_S^i(\psi)$, then $\mathrm{DOM}_S^{i+1}(\psi)$ can be expressed by

$$\bigcup_{i \in ar(\psi)} (\pi_2(\alpha^i(R_i)) \cup \cdots \cup \pi_{i+1}(\alpha^i(R_i)))$$

since $ar(\psi)$ is a finite set. $\mathrm{DOM}_S(\psi)$ can then be expressed as

$$\bigcup_{0 \leq i \leq d(\psi)+1} R_i$$

since $d(\psi)$ is finite.

The proof is then similar to that presented in [Ull89]. One proves by induction on the number of logical operators in subformulas ω of ψ that if ω has free variables $y_1, \cdots, y_k$, then

$$\mathrm{DOM}_S(\psi)^k \cap \{y_1, \cdots, y_k | \omega(y_1, \cdots, y_k)\}$$

has an equivalent expression in E-relational algebra. In particular, we shall have such an expression for $\omega = \psi$, and since ψ is strongly safe, intersection with $\mathrm{DOM}_S(\psi)^k$ will not change the denoted relation.

We show how variables in predicate names can be handled using E-relational algebra; the remaining logical operators are handled as for domain relational calculus in [Ull89]. We only need to modify the base case. Let the atom in question be

$$X_{i_0}(X_{i_1}, \cdots, X_{i_k}).$$

Then the E-relational algebra equivalent is of the form

$$\pi_{j_1+1} \cdots \pi_{j_l+1}(\sigma_F(\alpha^k(\mathrm{DOM}_S(\psi))))$$

where equality conditions in F represent repeated variables, and $\{X_{i_{j_1}}, \cdots, X_{i_{j_k}}\}$ are the unique variables among $X_{i_1}, \cdots, X_{i_k}$. ∎

Lemma 4.4: Every weakly safe relational HiLog expression has an equivalent expression in T-relational algebra.
Proof: (Sketch) First we observe that $\mathrm{DOM}_W(\psi)$ can be expressed in T-relational algebra as

$$\bigcup_{i \in ar(\psi)} (\rho_i \cup \pi_2(\alpha^i(\rho_i)) \cup \cdots \cup \pi_{i+1}(\alpha^i(\rho_i)))$$

since $ar(\psi)$ is a finite set.

The proof is then similar to that of Lemma 4.3 except that we substitute $\mathrm{DOM}_W(\psi)$ for $\mathrm{DOM}_S(\psi)$. ∎

We combine the above results to obtain the following theorems.

Theorem 4.5: The E-relational algebra and the strongly safe expressions of relational HiLog express the same set of queries. ∎

Theorem 4.6: The T-relational algebra and the weakly safe expressions of relational HiLog express the same set of queries. ∎

[5] The formula is not strongly safe, as X is not "reachable" from any fixed relation mentioned in the formula.

4.1 More Expressive Power?

One important question is "Does E-relational algebra or T-relational algebra yield a higher expressive power than standard relational algebra?" A similar question can be asked of the calculus languages. Clearly both extensions of relational algebra can simulate relational algebra by restriction.

The answer to this question depends on the type of transformation one is allowed to employ in showing that one formalism can simulate another. We show below that if the simulation is required to be *relation preserving* then we do get higher expressive power. A simulation is relation preserving if the relations being simulated are exactly the relations of the simulating formalism. In other words, the two formalisms are operating *on the same database.* Note that each of the simulations described above is relation preserving. Further, under this restriction we can show that T-relational algebra has higher expressive power than E-relational algebra.

On the other hand, if the simulation is not required to be relation preserving then we can show that standard relational algebra, E-relational algebra and T-relational algebra all have equal expressive power, as implied by the following theorem.

Theorem 4.7: For every database D there exists a database D' such that for every T-relational algebra expression F on D there is an equivalent expression F' on D' in standard relational algebra.
Proof: (Sketch) For each $k > 1$, construct a "universal relation" u_k of arity k. For each tuple $r(a_1, \cdots, a_n)$ in D include a tuple $u_{n+1}(r, a_1, \cdots, a_n)$ in D'. Selection, projection, cross-product, union and difference can be simulated in the obvious way. The expression $\alpha^i(R)$ on D can be expressed as

$$R \bowtie u_{i+1}$$

where the join is on the first column of u_{i+1}. ρ_i can be expressed as

$$\pi_1(u_{i+1}).$$

∎

Consequently, a similar equivalence holds for the calculus languages.

Relation-preservation is an important aspect of a simulation. If the set of relations is changed then information about functional dependencies and normal forms is lost. The simulation of Theorem 4.7 constructs "generic" relations about which no useful information can be gained. A real-world database system with generic relations would be impossible to manage.

Hence it is natural to require that relation-preservation be a *requirement* of a simulation. Theorem 4.7 should not be taken too seriously in light of the following result.

Theorem 4.8: There are expressions in the E-relational algebra not expressible (on the same database) by the standard relational algebra. Also, there are expressions in the T-relational algebra not expressible by the E-relational algebra.
Proof: (Sketch) $\alpha^i(R)$ cannot be expressed in standard relational algebra since the only possible way to reference relations in standard relational algebra is by explicit naming.

ρ_i cannot be expressed by the E-relational algebra, since the only ways to reference relations in E-relational algebra are by explicit naming or by using the value of an attribute of another relation as the name. Thus the E-relational algebra cannot in general construct the set of all nonempty relations of a given arity. ∎

Incidentally, the expansion operators cannot be expressed in terms of the totality operators and the standard relational algebra operators, and thus the expansion and totality operators are independent of one another.

5 Other Issues

5.1 Optimization

The expansion operators $\alpha^i(R)$ effectively form the union of a set of relations whose names are in R. Certainly any selections on the attribute R can be pushed inside the expansion operator.

While we may not know the names of the relations to be expanded when the query is initially posed, they do become available once R has been evaluated. Rather than actually computing $\alpha^i(R)$ immediately, we could choose to represent it in an intermediate form as the union of relations as in Definition 2.1. At this point, selections and projections applied to $\alpha^i(R)$ could be pushed inside the union. This strategy will reduce the number of tuples in the relations that will be unioned.

5.2 Schema and Functional Dependencies

On the surface, the extension of relational algebra and relational calculus to allow relation arguments to reference other relations does not impact upon the notion of the database scheme. However, the extra modelling capabilities available with such an algebra of calculus leads to applications in which the creation of new relations may be relatively frequent.

For example, one may have a large and changing inventory of devices in some application. Suppose that it is important to know the part-subpart hierarchy for each such device. Then we may choose to model the application using one "master" relation that identifies each device (given by some unique device number, say) with the relation representing its part hierarchy. The problem now faced is that every time a new device is added to the inventory, a new relation must be created for the corresponding part-subpart hierarchy.[6]

When relations are added or deleted frequently, then the assumption that the database scheme evolves slowly over time becomes suspect. A solution to this problem is to apply a typing discipline to arguments that are intended to be used as relation names. For example, it we may label the second argument of the master relation mentioned above as of type "binary relation in which the functional dependency $B \rightarrow A$ holds." By typing in this way we can be sure that newly created relations are in an appropriate normal form. Semantic constraints along these lines have been outlined in [SS77].

5.3 Deductive Databases

HiLog is useful in deductive databases for the same reasons that it is beneficial as part of a relational calculus. Even without function symbols, HiLog can be used to model sets and hierarchies without significant additional complexity. While the syntax may suggest that HiLog is second order, in a formal sense the semantics of HiLog programs is first order [CKW89].

On the other hand, one may ask when a deductive database based on HiLog (or relational HiLog) is *safe*, or *domain independent* in an analogous way to relational calculus. In general, the problem is undecidable [DiP69].

In fact, there is a stronger condition, called *preservation under extensions*, that effectively requires that the answer to a query (according to a given semantics) doesn't change when new relations sharing no symbols with the original database are added to the database [Ros91]. In [Ros91] it is shown that, while preservation under extensions and domain independence coincide for first-order deductive databases, the former is stronger for deductive databases based on HiLog. A sufficient condition for preservation under extensions, called *range restrictedness* for HiLog programs, has been proposed in [Ros91].

[6] This is a better solution than using one relation, namely the union of the part-subpart hierarchies with the part number as an extra argument. Space is saved by having to represent one less argument. In addition, if some new device is constructed as the union of smaller devices then we simply store all names of the part hierarchies for the subdevices in the master relation with the device number.

It is not difficult to show that strongly safe relational HiLog queries are preserved under extensions, while the class of weakly safe queries is not.

The translation of logical rules into a version of relational algebra for query evaluation could make use of the expansion operators defined here, and apply the optimizations discussed in Section 5.1. HiLog is currently being used as the basis for the NAIL deductive database system at Stanford University [PDR91].

5.4 Other features

In practice, query languages need additional features, such as aggregation. In [Klu82], Klug demonstrates an equivalence between versions of relational algebra and relational calculus that have been extended with aggregate operators. We conjecture that a similar equivalence could be proved for versions of E-relational algebra and T-relational algebra and relational HiLog that are extended with aggregate operators.

6 Conclusions

We have examined relational databases in which relation names may appear as arguments of other relations. Values in tuples may be used to reference the relation so named. We have proposed successive extensions of relational algebra to what we call E-relational algebra and T-relational algebra. We have also defined a calculus based on the logic HiLog for querying the database; we call this calculus relational HiLog.

We have proven the equivalence of our E-relational algebra an a class of "strongly safe" relational HiLog expressions. We have also proven the equivalence of our T-relational algebra and a class of "weakly safe" relational HiLog expressions. For a given database, T-relational algebra is more expressive than E-relational algebra, which is more expressive than standard relational algebra.

The extension that we have outlined requires only a very minor modification to existing relational systems, and provides more natural ways to model data. The expansion operator can be easily implemented, and it can be optimized by pushing selections and projections. Hence we feel that at least the E-relational algebra (or the strongly safe fragment of relational HiLog) should be expressible within modern query languages.

We have only looked at query languages here, and not data manipulation languages for updates and so on. One issue that needs to be addressed in a data manipulation language is referential integrity. If the name of a relation is modified, then appropriate tuples in various relations will have to be correspondingly modified. Semantic constraints along these lines are discussed in [SS77].

One of the ways we would like to generalize the methods presented here is to make the whole formalism less dependent on a relation's arity. For example, in the relational HiLog formula $p(X) \wedge X(Y, Z)$ it is assumed that X is used as a binary relation if it is mentioned in p. How might we be able to keep names of relations of several arities in p and access the relations with the appropriate arity? How would we name the attributes?

References

[C+76] D. D. Chamberlin et al. SEQUEL 2: A unified approach to data definition, manipulation and control. *IBM Journal of Research and Develipment*, 20(6):560–575, 1976.

[CKW89] W. Chen, M. Kifer, and D. S. Warren. HiLog: A first order semantics for higher-order logic programming constructs. In *Proc. North American Logic Programming Conference*, 1989.

[Cod70] E. F. Codd. A relational model for large shared data banks. *C. ACM*, 13(6):377–387, 1970.

[Cod72] E. F. Codd. Relational completeness of data base sublanguages. *Data Base Systems*, pages 65–98, 1972.

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

[Klu82] A. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *J. ACM*, 29(3):699–717, 1982.

[PDR91] G. Phipps, M. Derr, and K. A. Ross. Glue-Nail: A deductive database system. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, 1991.

[Ros91] K. A. Ross. On negation in HiLog. In *Proceedings of the Tenth ACM Symposium on Principles of Database Systems*, 1991. Full version submitted for journal publication.

[SS77] J. M. Smith and D. C. P. Smith. Database abstractions: Aggregation and generalization. *ACM Transactions on Database Systems*, 2(2):105–133, 1977.

[Ull89] J. D. Ullman. *Principles of Database and Knowledge Base Systems*. Computer Science Press, Rockville, MD, 1989. (Two volumes).