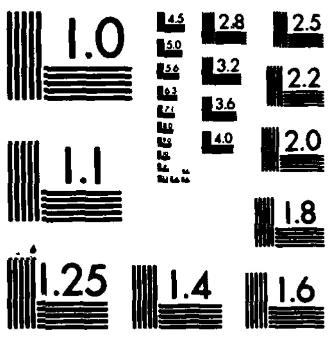
AD-A141 130   A NEW APPROACH TO DATABASE LOGIC(U) STANFORD UNIV CA   1/1
DEPT OF COMPUTER SCIENCE  G M KUPER ET AL. 1984
AFOSR-TR-84-0361 AFOSR-80-0212

UNCLASSIFIED   F/G 12/1   NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

# A New Approach to Database Logic

GABRIEL M. KUPER†
Stanford University
Stanford, California.

MOSHE Y. VARDI‡
IBM Research Laboratory
San Jose, California.

## 1 Introduction

In this paper we propose a mathematical framework for unifying and generalizing the three principal data models, i.e., the relational, hierarchical and network models ([U]). Until recently most work on database theory has focussed on the relational model ([C1]), mainly due to its elegance and mathematical simplicity compared to the other models. Some of this work has pointed out various disadvantages of the relational model, among them its lack of semantics ([C2], [HM], [SmSm]) and the fact that it forces the data to have a flat structure that the real data does not always have.

Several recent papers have addressed this problem by trying to find a more general mathematical framework. Specifically, Jacobs [J] describes "database logic," a mathematical model for databases that claims to generalize all three principal data models. Also, Hull and Yap [HY] describe the "format model." In their model, they view database schemes as trees, where each leaf represents data, and each internal node represents some connection between the data.

Both these models are unsatisfactory in their ability to restructure data, i.e., the ability to query the database. While Hull and Yap ignore the issue of a data manipulation language, Jacobs' treatment is an overkill—his query language enables one to write noncomputable queries [V].

Furthermore, both approaches fail to model a significant aspect of hierarchical and network database management systems, which is the ability to use *virtual records*. Virtual records are essentially pointers to physical records, and they are used to avoid redundancy in the database [U]. Note that virtual records introduce cyclicity not only in the schema level but also at the instance level.

In the model we propose here a database scheme is an arbitrary directed graph. As in the format model, leaves (i.e., nodes with no outgoing edges) represent data, and internal nodes repre-

† Work supported by AFOSR grant 80-0212

‡ This work was done while this author was at Stanford University and supported by a Weismann Fellowship and AFOSR grant 80-0212.

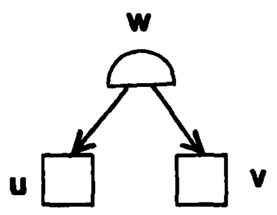


*Figure 2* Format Representation of Fig. 1.

**Example 1.** Assume we are given the *PERSON–PARENT* relation shown in Fig. 1. We can represent the structure of this relation in the format model by the format in Fig. 2. This format has two nodes $u$ and $v$ of type $\square$ that correspond to the attributes of the relation, and one node $w$ of type ⌓ that connects the pairs of related attributes.

An instance of this format will be an assignment of values to each node., as follows. We shall use the notation $I(u)$ to mean the set of values assigned to the node $u$ by the instance $I$. We could just take as the instance of a node a set of elements from the underlying domain, or tuples or sets taken from the instance of the node's successor. If we were to use this approach, we would not be able to deal with cycles in the format, and even if the format were acyclic, we would lose the ability to represent pointers to other nodes in our model, since the data would be represented explicitly at each node. What we do instead is have the instance of each node consist of a set of *l-values*, with corresponding *r-values*.

Intuitively, r-values constitute the data space, and the l-values constitute the address space. The instance of a node consists of set of l-values, with an r-value assigned to each of them. Formally, the l-values are elements of a fixed set $L$ (usually taken to be the natural numbers). We require that the instances of different nodes be disjoint. We also have a function $r$ on $L$, that assigns *r-values* to these l-values, and we require that the r-values be of the correct form, depending on the type of the node.

**Definition 2.** *An instance of a schema $S = \langle G, \mu \rangle$ consists of a mapping $I$ from $V$ to $\mathcal{P}^{fin}(L)$ (all finite subsets of $L$), and a mapping $r$ from $\bigcup_{v \in V} I(v)$; $r$ maps l-values to their r-values. If $v \neq w$, then $I(v)$ and $I(w)$ must be disjoint. For each node $v$ in $G$, $I(v)$ must satisfy*

(*1*) *If $\mu(v) = \square$, then for each $l \in I(v)$, $r(l)$ must be in $D$.*

(*2*) *If $\mu(v) =$ ⌓ and $v_1, \ldots, v_n$ are the successors of $v$, then for any $l \in I(v)$, $r(l)$ must be a tuple $(l_1, \ldots, l_n)$, such that for each $i$ between 1 and $n$, $l_i$ is an element of $I(v_i)$. An l-value in $I(v_i)$ can appear in any number of tuples, including none of them.*

(*3*) *If $\mu(v) = \bigcirc$ and $\tilde{v}$ is $v$'s successor, then $r(l)$ must be a subset of $I(\tilde{v})$.*

*If $l$ is an l-value, $r(l)$ is called the r-value of $l$.*

| $I(u)$ | | $I(v)$ | | $I(w)$ | |
|---|---|---|---|---|---|
| $l$ | $r(l)$ | $l$ | $r(l)$ | $l$ | $r(l)$ |
| 1 | Rehoboam | 4 | Solomon | 8 | (1,4) |
| 2 | Solomon | 5 | David | 9 | (2,5) |
| 3 | David | 6 | Batsheba | 10 | (2,6) |
| | | 7 | Jesse | 11 | (3,7) |

*Figure 3* Instance for the First Example

sent connections between the data. While it is not hard to model cyclicity at the schema level, it is not quite apparent how to do it at the instance level without running into cyclic definitions. Our solution is to keep the obvious distinction between memory locations and their content. Thus, instances in our model consists of *r-values*, which constitutes the data space, and *l-values*, which constitutes the address space. This mechanism enables us to give semantics to instances in a well-defined way.

A data model consists of several components (see [TL]). The first is the database structure mentioned above which describes the static portion of the database. The second component is a way to specify integrity constraints on the database, that restrict the allowed instances of the schema. We shall describe a logic in which integrity constraints can be specified. Unlike Jacobs' logic, our logic is effective. That is, given a database and a sentence in the logic, one can test effectively whether the sentence is true in the database or not.

The third component will be a way to restructure data, in order to describe user views, query languages, etc. We describe two such mechanisms, a logical, i.e., non-procedural, query language and an algebra, i.e., procedural, query language that are analogous to Codd's tuple calculus and relational algebra, and we prove them equivalent. These languages have a novel feature: not only can they access a non-flat data structure, i.e., a hierarchy, but the answers they produce do not have to be flat either. Thus, the language really does have the ability to restructure data and not only to retrieve it.

## 2 The Format Model

In our model, a schema is an arbitrary directed graph, with a type associated with each node. These types can be as follows.

(1) *Basic type*, written □. Nodes of this type contain the data stored in the database.

(2) *Composition*, written ▯. Nodes of this type contain tuples whose components are taken from the successors of the node.

(3) *Collection*, written ○. Nodes of this type contain sets, all of whose elements are taken from the node's successor.

Formally,

**Definition 1.** *A schema is a directed graph $G$, together with a function $\mu$ that assigns a type to each node of $G$. $\mu$ is a function from $V$, the set of nodes of $G$, to the set $\{$□, ○, ▯$\}$. $\mu(v)$ can be □ only when $v$ has no successors; It can be ▯ only when $v$ has at least one successor; ○ only when $v$ has exactly one successor.*

For each node $v$ of type ▯ we have an ordering of its successors, so that we can refer uniquely to "the $k^{th}$ successor" of $v$. Note that we do not have pointer types explicitly. However if we wanted them in the model, we could describe them as ▯-nodes with exactly one successor.

| *PERSON* | *PARENT* |
|---|---|
| Rehoboam | Solomon |
| Solomon | David |
| Solomon | Batsheba |
| David | Jesse |

*Figure 1* *PERSON*-*PARENT* Relation.

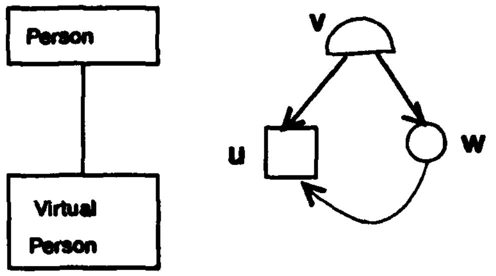


*Figure 4* Hierarchy for the Genealogy.

$I(u)$

| $l$ | $r(l)$ |
|---|---|
| 1 | Rehoboam |
| 2 | Solomon |
| 3 | David |
| 4 | Batsheba |
| 5 | Jesse |

$I(w)$

| $l$ | $r(l)$ |
|---|---|
| 6 | $\{2\}$ |
| 7 | $\{3,4\}$ |
| 8 | $\{5\}$ |

$I(v)$

| $l$ | $r(l)$ |
|---|---|
| 9 | $(1,6)$ |
| 10 | $(2,7)$ |
| 11 | $(3,8)$ |

*Figure 5* Instance of the Format in Example 2.

In Fig. 3 we show an instance of the format in Fig. 2 corresponding to the data in Fig. 1.

**Example 2.** We could also be given the genealogy as a hierarchy, as shown in Fig. 4. This could be represented in the format model as the format in Fig. 4, with the instance in Fig. 5. In general, given a hierarchy, we can convert it to a hierarchy as follows. Let $R_1, \ldots, R_n$ be the nodes in the hierarchy (the logical record types). For each $R_i$ we have a corresponding ⌓-node $v_i$ in the format. For each field of the logical record type $R_i$, $v_i$ has one successor of type □. The links are represented as follows. If $L_i$ is a link in the hierarchy, with owner $R_j$ and member $R_k$, we have in the format a node $w_i$ of type ○, that is a successor of $R_k$, and whose successor is $R_j$.

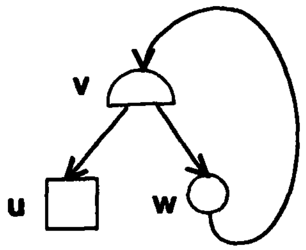


*Figure 6* Another Representation of the Genealogy.

$I(u)$

| $l$ | $r(l)$ |
|---|---|
| 1 | Rehoboam |
| 2 | Solomon |
| 3 | David |
| 4 | Batsheba |
| 5 | Jesse |

$I(v)$

| $l$ | $r(l)$ |
|---|---|
| 6 | $(1,11)$ |
| 7 | $(2,12)$ |
| 8 | $(3,13)$ |
| 9 | $(4,14)$ |
| 10 | $(5,14)$ |

$I(w)$

| $l$ | $r(l)$ |
|---|---|
| 11 | $\{7\}$ |
| 12 | $\{8,9\}$ |
| 13 | $\{10\}$ |
| 14 | $\emptyset$ |

*Figure 7* Instance of the Format in the Figure 6.

**Example 3.** We can also use the format model to represent data structured in ways that do not correspond to any of the standard data models. For example, we could represent the genealogy by the format in Fig. 6, and the instance in Fig. 7.

## 3 Logic

We define a calculus on formats in two stages. In this section we define a logic on formats. Then in the next section, we use this logic to define queries on formats. These queries will correspond to tuple calculus expressions in the relational model. We can also use the calculus to describe integrity constraints in the database.

Each variable in our logic has a fixed sort, where the sorts are nodes in the graph. The sorts restrict the possible values the variable may take. For example, if $x$ is a variable of sort $v$, $x$ can take only values in $I(v)$. In future, we will usually subscript the variable with its sort, e.g., $x_v$. Though the values of variables are always l-values, we shall say "the l-value of $x_v$" when we mean the value of $x_v$, and "the r-value of $x_v$" when we mean the r-value of the value of $x_v$.

**Definition 3.** *An atomic formula is one of*

(*1*) $x_v\ \pi_t\ y_w$, *meaning that the l-value of $x_v$ is the $t^{th}$ component of the r-value of $y_w$.*

(*2*) $x_v \in y_w$, *meaning that the l-value of $x_v$ is a member of the r-value of $y_w$.*

(*3*) $x_v =_l y_w$, *meaning that the l-values of $x_v$ and $y_w$ are equal.*

(*4*) $x_v =_r y_w$, *meaning that the r-values of $x_v$ and $y_w$ are equal.*

(*5*) $x_v =_r d$, *where $d$ is an element of the domain $D$, meaning that the r-value of $x_v$ is $d$.*

We then define well formed formulas in the usual way. $\models_I \phi(l_1,\ldots,l_n)$ will mean that $\phi$ is satisfied by $l_1,\ldots,l_n$ in the instance $I$. This is defined as follows.

**Definition 4.** *Let $\phi(x^1_{v_1},\ldots,x^n_{v_n})$ be a formula with free variables $x^1_{v_1},\ldots,x^n_{v_n}$. Let $l_1,\ldots,l_n$ be l-values, where for each $i$, $l_i \in I(v_i)$. Then $\models_I \phi(l_1,\ldots,l_n)$ is defined by induction on the size of $\phi$, as follows.*

(*1a*) *If $\phi$ is $x^i_v\ \pi_t\ y^j_w$, then $\models(x^i_v\ \pi_t\ x^j_w)(l_1,\ldots,l_n)$ iff $w$ is of type $\square$ with at least $t$ successors, and $l_i = \Pi_t(l_j)$.*

(*1b*) *If $\phi$ is $x^i_v \in x^j_w$, then $\models(x^i_v \in x^j_w)(l_1,\ldots,l_n)$ iff $w$ is of type $\bigcirc$ and $l_i$ is an element of $r(l_j)$.*

(*1c*) *If $\phi$ is $x^i_v =_l x^j_w$, then $\models(x^i_v =_l x^j_w)(l_1,\ldots,l_n)$ iff $l_i = l_j$.*

(*1d*) *If $\phi$ is $x^i_v =_r x^j_w$, then $\models(x^i_v =_r x^j_w)(l_1,\ldots,l_n)$ iff $r(l_i) = r(l_j)$.*

(*1e*) *If $\phi$ is $x^i_v =_r d$, where $d$ is an element of $D$, then $\models(x^i_v =_r d)(l_1,\ldots,l_n)$ iff $r(l_i) = d$.*

(*2*) *If $\phi$ is $\phi_1 \vee \phi_2$ or $\neg\phi_1$, then the definition is the usual one.*

(*3*) *If $\phi$ is a formula with free variables $x^1_{v_1},\ldots,x^n_{v_n}$, and $y_w$, then*

$$\models ((\exists y_w)\phi)(l_1,\ldots,l_n)$$

*iff there is an $l$ in $I(w)$ such that*

$$\models \phi(l_1,\ldots,l_n,l).$$

**Example 4.** The formula $x_u\ \pi_1\ y_v$ says that the l-value of $x_u$ is equal to the first component of the r-value of $y_v$. It is satisfied in the instance of Example 3 by the $(x_u, y_v)$ pairs $(1,7)$, $(2,8)$, $(3,9)$, $(4,10)$, and $(5,11)$.

**Lemma 1.** *If $\phi(x^1_{v_1},\ldots,x^n_{v_n})$ is a formula with free variables $x^1_{v_1},\ldots,x^n_{v_n}$, and $l_1,\ldots,l_n$ are l-values, where for each $i$, $l_i \in I(v_i)$, then $\models_I \phi(l_1,\ldots,l_n)$ can be determined effectively.*

*Proof:* This is shown by induction on the size of the formula. For atomic formulas testing for satisfaction is straightforward. Testing for disjunction and negation is also clearly effective, and the result for quantification follows from the fact that all instances are finite. ∎

## 4 Queries

In the relational model the result of a query is a relation. We might therefore in analogy expect that the result of a query on the format model will be a format, i.e., a schema and an instance of it. This approach generalizes the relational algebra approach and may also suggest query languages for the other data models.

We modify this approach a bit by not requiring that the query's schema be an independent schema, but we allow the successors of nodes in the query to be nodes in the database. One reason for this is that we may want our answer to have references to the database rather than copies of large structures. Another reason is to simplify the definitions of the algebraic operations. We shall want each algebraic operation to be the result of some query, but on the other hand we would like to be able to simulate an arbitrary "safe" query by a sequence of algebraic operations, and if each operation had to create a completely new format, the definitions would be unnecessarily complicated. Notice that if the query were a "normal query," i.e., one which created a completely new structure, the corresponding algebraic operations would first copy the required nodes, and then would combine l-values only from these new nodes.

The natural thing to do now, using the analogy with the tuple calculus in the relational model, is to take some formula $\phi$ and let the instance be those things satisfying it.

There are two problems with this approach. The first is that the queries cannot build the l-values by themselves—such a formula just says when a given instance satisfies it, but gives no way to construct such an instance. One solution is to prevent the query from referring directly to l-values, and allow them to be referred to only by their r-values. We could then find all possible r-values than could appear in the result, assign them arbitrary l-values, and try to show that the result is unique up to isomorphism.

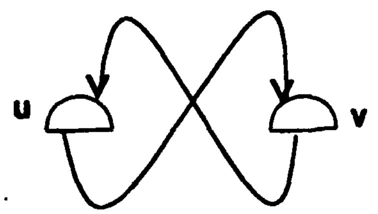


*Figure 8*

$I_1(u)$

| $l$ | $r(l)$ |
|---|---|
| 1 | (3) |
| 2 | (4) |

$I_1(v)$

| $l$ | $r(l)$ |
|---|---|
| 3 | (1) |
| 4 | (2) |

*Figure 9* A Possible Result of the Query

$I_2(u)$

| $l$ | $r(l)$ |
|---|---|
| 1 | (3) |
| 2 | (4) |

$I_2(v)$

| $l$ | $r(l)$ |
|---|---|
| 3 | (2) |
| 4 | (1) |

*Figure 10* Another Possible Result of the Query.

Another problem with this approach is dealing with cyclicity. We need the ability to refer directly to l-values in order to make use of the cyclicity, but even then the result of the query will not always be defined uniquely. For example, if the query schema is the format in Fig. 8, and the query just specifies that $I(u)$ and $I(v)$ each contain at least two different l-values. Then Fig. 9 and 10 shows two incomparable instances, and we have no way to choose between them. Our solution has been to restrict the queries to ones not containing cycles, while allowing cyclicity in the database, and allowing the query to refer explicitly to l-values in the database.

The formal definition of a query is

**Definition 5.** *Given a database schema $S = \langle G, \mu \rangle$, and an instance $I$, a query $Q = \langle S', \Phi \rangle$ on the database consists of*

*(1) A set of nodes and directed edges with types associated with each node, $S' = \langle G', \mu' \rangle$, such that*

*(a) $G'$ is a DAG, and edges can also connect nodes of $G'$ to nodes of $G$.*

*(b) $\langle G \cup G', \mu \cup \mu' \rangle$ is a schema, which we shall call $S^* = \langle G^*, \mu^* \rangle$.*

*(2) The fact that $G'$ is a DAG implies that there is an order $\prec$ on the nodes of $G'$, such that if $v$ and $w$ are nodes of $G'$ and $v$ is a successor of $w$, then $v \prec w$. Let $\prec$ be a fixed ordering of the nodes with this property.*

*(3) $\Phi$ is a set of formulae, one for each node $v$ of $G'$. The formula $\phi_v$ corresponding to $v$ must satisfy*

*(a) There is only one free variable in $\phi_v$, and it is of sort $v$.*

*(b) All other variables are bound, and their sorts are either nodes of the database, or are nodes that precede $v$ under $\prec$.*

Since the query is now acyclic, we can create the result of the query "bottom-up," i.e., we define the result of the query at each node in terms of the results at its successors. We define the result of the query by the following inductive construction. Assume that $I(w)$ has been defined for all nodes $w$ in the query that precede $v$ under $\prec$. We then say that $r$ is a *candidate r-value for v* if by setting $r(l) = r$, and letting $I(v)$ contain the single l-value $l$, we get $\models_I \phi_v(l)$ ($l$ is an arbitrary unused l-value). The construction of $I(v)$ is as follows. Let $R$ be the set of all candidate r-values for $v$. For each $r \in R$ arbitrarily select a different unused l-value $l_r$, set $r(l_r) = r$, and let $I(v)$ contain of all these l-values. Repeat this for each node $v$ in the order given by $\prec$.

We now give the formal definition of the result of a query. We start with the definition of candidate r-value.

**Definition 6.** *Let $\phi_v(x_v)$ be a formula with free variable $x_v$ and let $I$ be an instance of some of the nodes in the format, including the sorts of all the bound variables in $\phi_v$. Let $l$ be an l-value that is unused in $I$. We say that $r$ is a* **candidate r-value** *for $v$ if by setting $r(l) = r$, we get $\models_{I \cup \{l\}} \phi_v(l)$. Let $R$ be the set of all candidate r-values for $v$. For each $r \in R$, select a different unused l-value $l_r$ (the choice of l-values is arbitrary), and set $r(l_r) = r$. Then $[l \mid \phi_v(l)]$ is defined to be $\{l_r \mid r \in R\}$.*

**Lemma 2.** *In Definition 6, assume that $R$ is finite. Then if $I(v)$ is defined to be $[l \mid \phi_v(l)]$, the following hold.*

*(i) For each $l$ in $I(v)$, $\models_I \phi_v(l)$.*

*(ii) If we take different unused l-values in the definition, we get an isomorphic instance $I'$.*

*(iii) There are no two different l-values in $I(v)$, $l_1 \neq_l l_2$ with $r(l_1) = r(l_2)$.*

*(iv) $I(v)$ is maximal satisfying (i)–(iii).* ∎

**Definition 7.** *The result of the query is an instance $I^*$ of the schema $S^*$. It is defined by induction as follows. If $v$ is a node of the database, we define $I^*(v) = I(v)$. If $v$ is a node of the query, assume that we have already defined $I^*(w)$ for any node $w$ that precedes $v$ under $\prec$. Then $I^*(v)$ is defined as $[l \mid \phi_v(l)]$.*

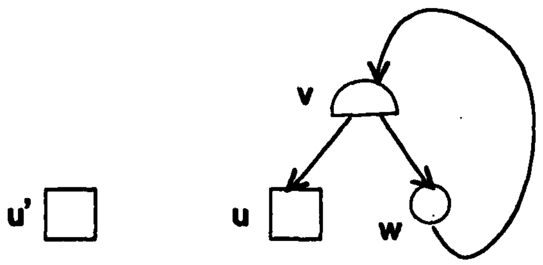


*Figure 11* Query on the Genealogy Database.

$I(u')$

| $l$ | $r(l)$ |
|---|---|
| 17 | Rehoboam |
| 18 | Solomon |
| 19 | David |
| 20 | Batsheba |
| 21 | Jesse |

*Figure 12* Possible Result of the Query.

**Example 5.** Assume that the database is the genealogy format of Fig. 6 with the instance of Fig. 7. The query will consist of the node $u'$ in Fig. 11, with formula $\phi_{u'}(x_{u'}) = (\exists y_u)(x_{u'} =_r y_u)$. In other words, we want $I(u')$ to be a copy of $I(u)$ (removing duplicated values, if $I(u)$ had any). To answer the query we do the following. First, take an unused l-value, say 17. Now look for all possible r-values $r$ (in this case, elements of $D$), such that if we set $r(17) = r$, and $I(u') = \{17\}$, we get $\models_I \phi_{u'}(17)$. The set $R$ of candidate r-values is $R$ = {Rehoboam, Solomon, David, Batsheba, Jesse}, and therefore the result of the query (up to isomorphism) is as shown in Fig. 12.

**Definition 8.** *A query $Q$ on a database with schema $S$ is safe if for every instance $I$ of $S$ the result of the query exists.*

The following lemma shows that to check if a query is safe, it suffices to check the results at the leaves.

**Lemma 3.** *A query $Q$ on database schema $S$ is safe iff for every instance $I$ of the database,. and for every node $v$ of the query of type $\square$, the set of candidate r-values for $v$ is finite.* ∎

**Lemma 4.** *Let $Q$ be a query on a database with schema $S$, and let $w_1, \ldots, w_n$ be the nodes in the database of type $\square$. $Q$ is safe iff if there is a finite set $\{d_1^*, \ldots, d_{k^*}^*\}$ of elements of $D$ such that for every instance $I$ of the database $S$, and for every node $v$ of type $\square$ in the query $Q$, all of the candidate r-values for $I(v)$ are either r-values of elements of the $I(w_i)$'s or are among the $d_j$'s.* ∎

The constants in the above lemma are those elements of $D$ that are mentioned in any of the formulas $\phi_v$.

## 5 Algebra

We now define an algebraic query language. This language is equivalent to the logical query language. That is, each logical query is equivalent to a logical query and vice versa.

The algebraic language consists of the following basic operations:

(1) $w \leftarrow v$ creates a new node $w$ of the same type as $v$, and with the same successors as $v$. $I(w)$ contains a copy each l-value in $I(v)$.

(2) $w \leftarrow \square(d)$ creates a node $w$ of type $\square$, that contains a single l-value, whose r-value is $d$.

**Example 6.** Let the database be the genealogy of Fig. 6 with the instance of Fig. 7. The operations $u' \leftarrow \square(u)$ and $v' \leftarrow$ ("Absalom") each add a new node $u'$ and $v'$ respectively to the database. Their instances are shown in Fig. 13 (a) and (b).

$I(u')$

| $l$ | $r(l)$ |
|---|---|
| 17 | Rehoboam |
| 18 | Solomon |
| 19 | David |
| 20 | Batsheba |
| 21 | Jesse |

$I(v')$

| $l$ | $r(l)$ |
|---|---|
| 17 | Absalom |

*Figure 13* Examples of (a) $u' \leftarrow u$ (b) $v' \leftarrow \square(d)$.

(3) $w \leftarrow \bigcirc(v)$ creates a new node $w$ of type $\bigcirc$ with successor $v$. $I(w)$ contains a copy of each possible subset of $I(v)$.

(4) $w \leftarrow$ ⌓$(v_1, \ldots, v_n)$ creates a new node $w$ of type ⌓, with successors $v_1, \ldots, v_n$. $I(w)$ contains all possible tuples with $i^{th}$ component in $I(v_i)$.

(5) If $v$ is a node of type ⌓ with $n$ successors, and $i \; \theta \; j$ is one of the relations $i \in j$, $i \; \Pi_l \; j$, $i =_l j$, $i =_r j$ and $i =_r d$, then $w \leftarrow \sigma_{i \, \theta \, j}(v)$ creates a node $w$ of the same type as $v$ and with the same successors, that contains a copy of each tuple from $I(v)$ whose $i^{th}$ and $j^{th}$ components are in the specified relation.

(6) If $v_1, \ldots, v_n$ are all of the same type and have the same successors, then $w \leftarrow \cup(v_1, \ldots, v_n)$ creates another similar node $w$ that contains a copy of each element that is in one of the $I(v_i)$'s.

(7) If $v_1$ and $v_2$ have the same type and the same successors. then $w \leftarrow v_1 - v_2$ creates another similar node $w$ that contains a copy of each element of $I(v_1)$ whose r-value is not the r-value of any element of $I(v_2)$.

(8) If $v$ is of type ⌓ with successors $v_1, \ldots, v_n$, and $S = \{s_1, \ldots, s_k\}$ is a subset of the set $\{1, \ldots, n\}$, then $u \leftarrow \Pi_S(v)$ creates a new node $w$ of type ⌓ with successors $v_{s_1}, \ldots, v_{s_k}$, that contains all projections of tuples in $I(v)$ onto these components.

(9) If $v$ is of type ⌓ and has exactly one successor, $\tilde{v}$, then $w \leftarrow \tilde{\Pi}(v)$ creates a new node $w$ similar to $\tilde{v}$, that contains a copy of each element of $I(\tilde{v})$ that is the component of some element of $I(v)$.

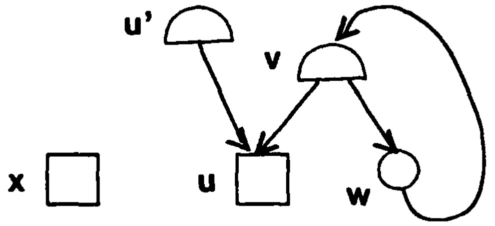


*Figure 14* Example of $\tilde{\Pi}$.

$I(u')$

| $l$ | $r(l)$ |
|---|---|
| 15 | (3) |
| 16 | (4) |

$I(x)$

| $l$ | $r(l)$ |
|---|---|
| 17 | David |
| 18 | Batsheba |

*Figure 15* Example of $\tilde{\Pi}(u')$.

**Example 7.** Suppose the database is the genealogy format with the extra node $u'$ shown in Fig. 14 and with the instances in Fig. 7 ($u$, $v$ and $w$) and Fig. 15 ($u'$). Then $x \leftarrow \tilde{\Pi}(u')$ creates the node $x$ in Fig. 14, with $I(x)$ as in Fig. 15.

**Main Theorem.** *The algebraic language and the logical language are equivalent, i.e., every algebraic query is equivalent to a safe logical query, and every safe logical query is equivalent to an algebraic query.*

**Outline of Proof.** The first direction of the theorem, that for each algebraic operation there is an equivalent query, is shown by creating, for each operation, a query that consists of one new node $w$ with formula $\phi_w$. This query will be safe and will have the same result as the corresponding algebraic operation. The details are fairly straightforward and will not be given here.

For the second part of the theorem, we are given a safe query, and we have to show that it can be simulated by a sequence of algebraic operations. Let $Q$ be the query. We shall construct the desired algebraic expression by induction on the nodes in the query using the order $\prec$. Assume therefore that we have an algebraic expression for each node that precedes a node $w$ in the query, and we now want to find an algebraic expression that constructs the node $w$. The formula corresponding to $w$ is $\phi_w(x_w)$. Let the bound variables of $\phi_w$ be $x^1_{v_1}, \ldots, x^n_{v_n}$ (where each variable is bound by exactly one quantifier). Since the query is safe, there is a set $\{d_1, \ldots, d_k\}$ of elements of $D$, such that each r-value of a node of type □ is either one of these constants, or is an r-value of some node in the database of type □.

Our first step is to create a node $\tilde{w}$ that represents the domain of $w$, i.e., all the r-values that elements of $I(w)$ could possibly have, if $\phi_w$ contained no restrictions apart from the safeness requirement given above. We define $\tilde{w}$ as follows.

(1) If $w$ is of type □, and $v_1, \ldots, v_n$ are all the nodes in the database of type □, then define $\tilde{w}$ as follows. (i) Let $w_i \leftarrow v_i$ for $1 \leq i \leq n$ (ii) Let $w_{n+i} \leftarrow □(d_i)$ for $1 \leq i \leq k$, where the $d_i$'s are the constants listed above. (iii) Let $\tilde{w} \leftarrow w_1 \cup \cdots \cup w_{m+k}$.

(2) If $w$ is of type ⬯ and its successors are $w_1, \ldots, w_k$, let $\tilde{w} \leftarrow w_1 \times \cdots \times w_k$.

(3) If $w$ is of type ○ and its successor is $u$, let $\tilde{w} \leftarrow ○(u)$.

We can then show:

**Lemma 5.** *Let $I(w)$ be the result of the given query at node $w$. Then every $l$ in $I(w)$ must satisfy $r(l) \in r[I(\tilde{w})]$.* ∎

Let the bound variables in $\phi_w$ be $x^1_{v_1}, \ldots, x^n_{v_n}$. Let

$$u \leftarrow v_1 \times \cdots \times v_n \times \tilde{w},$$

and "label" each $v_i$ with the variable $x^i_{v_i}$. This enables us to distinguish between two copies of the same node that came from different variables. Also label $\tilde{w}$ with $x_w$. We define nodes $v_\psi$, for each well-formed subformula $\psi$ of $\phi_w$, by induction on the size of $\psi$, as follows.

(1) If $\psi$ is an atomic formula of the form $x^i_{v_i}\ \theta\ x^j_{v_j}$, let $v_\psi \leftarrow \sigma_{i\,\theta\,j}(u)$. If $\psi$ is of the form $x^i_{v_i}\ \theta\ x_w$, let $v_\psi \leftarrow \sigma_{i\,\theta\,n+1}(u)$, similarly for the cases where $\psi$ is of the form $x_w\ \theta\ x^i_{v_i}$ and $x_w\ \theta\ x_w$. In each case $\psi$ has the same successors as $u$.

(2) If $\psi$ is $\psi_1 \vee \psi_2$, $v_{\psi_1}$ and $v_{\psi_2}$ may have different successors. We shall show below that $\tilde{w}$ is always a successor of each $v_\psi$. Let the common successors of $v_{\psi_1}$ and $v_{\psi_2}$ be labeled $x^{s_1}_{v_{s_1}}, \ldots, x^{s_k}_{v_{s_k}}$ and $x_w$. Let $v'_{\psi_1} \leftarrow \Pi_{S_1}(v_{\psi_1})$ and $v'_{\psi_2} \leftarrow \Pi_{S_2}(v_{\psi_2})$, where $S_1$ and $S_2$ are the numbers of the components of $v_{\psi_1}$ and $v_{\psi_2}$ corresponding to the common successors of $v_{\psi_1}$ and $v_{\psi_2}$. Then let $v_\psi \leftarrow v'_{\psi_1} \cup v'_{\psi_2}$.

(3) If $\psi$ is $\neg\psi'$, let $x^{s_1}_{v_{s_1}}, \ldots, x^{s_k}_{v_{s_k}}$ and $x_w$ be the labels of the successors of $v_{\psi'}$. Let

$$u_{\psi'} \leftarrow \Pi_{\{s_1,\ldots,s_k,n+1\}}(u),$$

and $v_\psi \leftarrow u_{\psi'} - v_{\psi'}$.

(4) If $\psi$ is $(\exists x^i_{v_i})(\psi')$ and $x^{s_1}_{v_{s_1}}, \ldots, x^{s_k}_{v_{s_k}}$, $x_w$ are the labels of the successors of $v_{\psi'}$, assume that $v_i$ is the $j^{\text{th}}$ component of $v_{\psi'}$. Let

$$v_\psi \leftarrow \Pi_{\{1,\ldots,k+1\}-\{j\}}(v_{\psi'}).$$

It can then be shown that:

**Lemma 6.** *For each subformula $\psi$ of $\phi = \phi_v$,*

*(1) $v_\psi$ is of type ⬯.*

*(2) The successors of $v_\psi$ are $\tilde{w}$ and the sorts of the all the bound variables that appear in $v_\phi$, except for those that are bound by a quantifier in $\psi$.*

*(3) Assume that the successors of $v_\psi$ are labelled $x^{s_1}_{v_{s_1}}, \ldots, x^{s_k}_{v_{s_k}}$ and $x_w$. Then*

$$\begin{aligned} I(v_\psi) = [l \mid\ & r(l) = (l_1, \ldots, l_{k+1}) \\ & \wedge \models_I \psi(l_1, \ldots, l_{k+1}) \\ & \wedge (l_1, \ldots, l_{k+1}) \in r[\Pi_{\{i_1,\ldots,i_{k+1}\}}(I(u))]], \end{aligned}$$

## REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED | 1b. RESTRICTIVE MARKINGS |
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited. |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S) | 5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 84-0361 |
| 6a. NAME OF PERFORMING ORGANIZATION: Stanford University; 6b. OFFICE SYMBOL (If applicable) | 7a. NAME OF MONITORING ORGANIZATION: Air Force Office of Scientific Research |
| 6c. ADDRESS (City, State and ZIP Code): Computer Science Department, Stanford CA 94305 | 7b. ADDRESS (City, State and ZIP Code): Directorate of Mathematical & Information Sciences, Bolling AFB DC 20332 |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION: AFOSR; 8b. OFFICE SYMBOL (If applicable): NM | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: AFOSR-80-0212 |
| 8c. ADDRESS (City, State and ZIP Code): Bolling AFB DC 20332 | 10. SOURCE OF FUNDING NOS.: PROGRAM ELEMENT NO. 61102F; PROJECT NO. 2304; TASK NO. A7; WORK UNIT NO. |
| 11. TITLE (Include Security Classification): A NEW APPROACH TO DATABASE LOGIC | |

12. PERSONAL AUTHOR(S): Gabriel M. Kuper and Moshe Y. Vardi*

| | | | |
|---|---|---|---|
| 13a. TYPE OF REPORT: Technical | 13b. TIME COVERED: FROM ____ TO ____ | 14. DATE OF REPORT (Yr., Mo., Day): 1984 | 15. PAGE COUNT: 11 |

16. SUPPLEMENTARY NOTATION: *IBM Research Laboratory, San Jose, California.

17. COSATI CODES

| FIELD | GROUP | SUB. GR. |
|---|---|---|
| | | |
| | | |

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number): 1984

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

In this paper the authors propose a mathematical framework for unifying and generalizing the three principal data models, i.e., the relational, hierarchical and network models. Until recently most work on database theory has focussed on the relational model, mainly due to its elegance and mathematical simplicity compared to the other models. Some of this work has pointed out various disadvantages of the relational model, among them its lack of semantics and the fact that it forces the data to have a flat structure that the real data does not always have.

Several recent papers have addressed this problem by trying to find a more general mathematical framework. Specifically, Jacobs describes "database logic," a mathematical model for databases that claims to generalize all three principal data models. Also, Hull and Yap describe the "format model." In their model, they view (CONTINUED)

| | |
|---|---|
| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐ | 21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL: Dr. Robert N. Buchal | 22b. TELEPHONE NUMBER (Include Area Code): (202) 767-4939; 22c. OFFICE SYMBOL: NM |

*where* $\psi = \psi(x^{s_1}_{v_{s_1}}, \ldots, x^{s_k}_{v_{s_k}}, x_w)$. ∎

To prove the theorem, apply Lemma 6 to the formula $\phi$. We get a $\bigcirc$-node $v_\phi$, whose only successor is $\tilde{w}$ (all other variables are bound in $\psi$), and

$$I(v_\phi) = [l \mid r(l) = (l') \wedge \models_I \phi(l') \wedge (l') \in r[\Pi_{\{n+1\}} I(u)]].$$

Since by Lemma 5, $\models_I \phi(l')$ implies that $l' \in I(\tilde{w})$, and each element of $I(\tilde{w})$ is the only component of the r-value of some element of $r[\Pi_{n+1}(I(u))]$, we have

$$I(v_\phi) = [l \mid r(l) = (l') \wedge \models_I \phi(l')].$$

Finally, we let $w \leftarrow \tilde{\Pi}(v_\phi)$, which gives us a node $w$ of the same type as $\tilde{w}$ that satisfies $I(w) = [l \mid \phi(l)]$. ∎

## 6 Acknowledgments

We are indebted to Jeff Ullman for some of the basic ideas underlying this work. We also would like to thank Dave Maier and Howard Trickey for helpful discussions and suggestions.

## References

[C1] Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Communications of the ACM*, **13**:6 (June 70), pp. 377–387.

[C2] Codd, E. F., "Extending the Database Relational Model to Capture More Meaning," *ACM Transactions on Database Systems* **4**:4 (Dec 79), pp. 397 434.

[HM] Hammer, M. and D. McLeod, "Database Description with SDM—A Semantic Database Model," *ACM Transactions on Database Systems* **3**:3 (Sep 78), pp. 201–222.

[HY] Hull, R. and C. K. Yap, "The Format Model: A Theory of Database Organization," *to appear in J. ACM.*

[J] Jacobs, B. E., "On Database Logic," J. ACM **29**:2 (1982), pp. 310-332.

[SmSm] Smith, J. M. and D. C. P. Smith, "Database Abstractions: Aggregation," *Communications of the ACM*, **20**:6 (1977), pp. 405-413.

[TL] Tsichritzis, D. C. and F. H. Lochovsky, "Data Models," *Prentice Hall*, Englewood Cliffs, NJ., 1982.

[U] Ullman, J. D., "Principles of Database Systems," *Computer Science Press*, Rockville, MD., 1982.

[V] Vardi, M. Y., "Review of [J]," *Zentralblatt für Mathematik—479.680061.*