LAURA:
A Formal Data Model
and her Logical Design Methodology

Robert Brown

The Database Design Group
956 Jackson Street
Mountain View, CA 94043 USA

D. Stott Parker, Jr. *

Computer Science Department
University of California
Los Angeles, California 90024 USA

ABSTRACT

This paper describes the LAURA data model and its use in the logical design methodology of the Database Design Group. LAURA provides simple formal constructs for the design of large databases, including generalization, abstraction, and categorization primitives, while avoiding limitations of existing formal models.

LAURA is based on functional connections between data, and has features of functional and binary data models, and semantic networks. A consequence of choosing a relatively abstract level for modeling is that important semantic information is available for use in 'normalization' which can be lost, for example, when schemas are translated to relational representations. In particular, use of the relational model inhibits proper modeling of different types of functions and abstractional concepts, and clouds the fact that eliminating *inherited properties* from a schema is a main concern of normalization (which is not at all evident when expressed in the language of functional dependencies).

This paper concentrates first on introducing LAURA, and second on describing the normalizing transforms used with the model. The approach developed here has been automated, and is being used successfully in the interactive design of large database systems.

* Work supported by NSF grant IST 80-12419.

1. Introduction

Currently, the problem of logical design of large databases is poorly understood, even by experts. Some formal foundation is needed to perform the job effectively, since without a formal basis, schemas and data models tend to degenerate into ambiguous or meaningless scribbles. Unfortunately, existing models can be of little help to the practicing database designer in developing formally-based design tools. Some designers have resorted to their own heuristic approaches, following intuition. Others have developed programs which require large clusters of Functional Dependencies to be entered. An alternative formal approach is needed, which is simple yet offers powerful modeling constructs.

This paper describes the logical data model, and logical design methodology, of Database Design Group (DBDG). The methodology in its entirety progresses through four phases, where the results of each phase are translated into progressively more specific models:

Phase 1. Entity Relationship Model
Phase 2. LAURA: 'Substrate' Model
Phase 3. Structural Model
Phase 4. DBMS/Physical Model

The Entity-Relationship and Structural Models are related to LAURA by direct translations. LAURA is of particular interest, and most of this paper is devoted to its description and its use in 'normalizing' schemas.

In the DBDG methodology, an initial conceptual model of the database is obtained iteratively. At each step of the refinement process, the database designer transcribes his understanding of the various user views required by the enterprise using a *diagramming language*. This encoded statement of the enterprise is fed to a program that produces simplified Entity-Relationship diagrams. These diagrams are discussed by the designer and members of the enterprise to reach improved understanding of how the database should be structured.

View merging software is then used to integrate the diagramming language representations of the views, producing an enterprise schema. When this schema is complete, it is changed to a LAURA representation. LAURA is graph-oriented, and can model entities, functional associations, inclusion relationships & existence constraints, and permissibility of null values in fields. All associations are functional in LAURA.

At this point 'normalizing' transforms may be applied to the LAURA schema. These transforms improve the schema, making either minor corrections to misspecifications (e.g., add existence constraints, merge identical but differently named attributes), or changes preserving logical connections in the schema.

The LAURA model is finally translated to a structural model that is similar to the relational model. We feel that significant benefit comes from doing 'normalizing' transforms on the LAURA graph model *before* semantic information is lost in this translation. Much work on normalization in the relational model takes on a new significance when semantics are considered in the process, and artifacts imposed by the relational model are removed.

2. Background

We assume the reader conversant with standard database terminology. We apologize only for the use of a few terms. The words 'object', 'association', and 'connection' are used initially here in an abstract sense. This usage conflicts with that of a number of workers in the database theory area.

Also, we use the word *attribute* here to mean a symbol in a fixed set

$$\Omega = \{A_1, \ldots, A_n\},$$

while 'property' is used to convey the more abstract notion of a 'characteristic of an entity'. As usual, each attribute A has a *domain* $dom(A)$, and sequences of attributes $S = A_{i_1}, \ldots, A_{i_n}$ have domain $dom(S) = dom(A_{i_1}) \times \cdots \times dom(A_{i_n})$.

Finally, we assume the reader to have a basic understanding of abstraction concepts, the binary model, and semantic networks. We develop all terminology needed here, but the reader may wish to review these subjects. Smith & Smith [SS1,SS2] give an excellent development of generalization and aggregation, extending earlier work by Quillian and others on semantic networks. Brachman's survey on semantic networks [B] is also recommended. Interesting treatments of the binary data model can be found in [A], [TL]. [HK] provides a good presentation of the functional model.

2.1. Connections in a Database

Let us begin with an intuitive development of *connections*. Suppose we are interested in the phone number of the manager of the database project. (The database is not working, say.) The 'connection' we use to find this information might be to find the phone number of the office whose employee is manager of the project. Using the abstraction primitives *is* and *has*, we employ

PROJECT has MGR is EMP has OFFICE has PHONE

Similarly we can express 'recursive' connections such as the salary of a particular employee's manager's manager:

EMP has MGR is EMP has MGR is EMP has SALARY.

We view *facts* in a database as connections among data objects. In the terminology above, connections might be made up of *is* and *has* relationships between objects. Formally, we model this by saying that if A and B are attributes, a *connection* $-\sigma-$ between them is a subset of $dom(A) \times dom(B)$. This definition extends to n-ary connections. This definition is purely extensional, but by considering classes of connections such as these we obtain an intensional definition.

A recurring concern in both the logical design and the querying of databases is the *combination* of connections. If A, B, C are attributes, and we have the connections σ, τ

$$A -\sigma- B -\tau- C$$

then their combination (which we may write as $\pi_{AC}(\sigma|\times|\tau)$, π denoting projection) gives a connection between $dom(A)$ and $dom(C)$. Generally speaking, any path of connections will define some connection. However, this combination may not be 'meaningful' (nontrivial). Let

$$dom(A)_\sigma, dom(B)_\sigma$$

denote the subsets of $dom(A)$, $dom(B)$ actually appearing in this binary relation. Then given the connections σ, τ among A, B, C above, the combination $\sigma|\times|\tau$ is meaningful only when σ and τ meet the weak *existence constraint*

$$dom(B)_\sigma \cap dom(B)_\tau \neq \phi.$$

Otherwise the combination is trivial.

Thus, by 'connections' we are really referring to project-join combinations. This is close to the Binary data model concept of association, which includes both navigational and existence semantics. Navigation is performed by combining connections. Existence semantics certify that these combinations are meaningful.

2.2. Functional Connections

Functions are common types of connections. They can be used to model more complex connections, such as many-many relationships and n-ary relationships. Let A, B be attributes. For a given functional connection $f: A \to B$, let us write

$dom(A)_f \subseteq dom(A)$ is the set on which f is defined
$dom(B)_f \subseteq dom(B)$ is the image of f

so that

$$f(dom(A)_f) = f(dom(A)) = dom(B)_f$$

Definition f is a *total function* if $dom(A)_f = dom(A)$; otherwise f is a *partial function*.

Definition f is *onto* if $dom(B)_f = dom(B)$.

Definition f is an *inclusion* if $dom(A)_f \subseteq dom(B)$ and f is the identity map on $dom(A)_f$.

If all connections are functional, functional composition yields a combination of connections. To determine the set of meaningful compositions, we can require specification, for each pair

of functions $A \xrightarrow{f} B \xrightarrow{g} C$, whether or not $dom(B)_f \cap dom(B)_g$ is empty. A more typical way to specify whether compositions are meaningful is to use *existence constraints*. Putting an existence constraint on a function $f: A \to B$ (i.e., asserting that whenever a is in $dom(A)$, there must exist some $b = f(a)$ in $dom(B)$) is identical to asserting that f be *total*.

Two important facts here are that *compositions of total functions are total*, and *compositions of onto functions are onto*. As a result, compositions of total functions are always 'meaningful' (nontrivial). This is not to say that composition of partial functions is not meaningful; composition of partial functions is just not *guaranteed* to be meaningful.

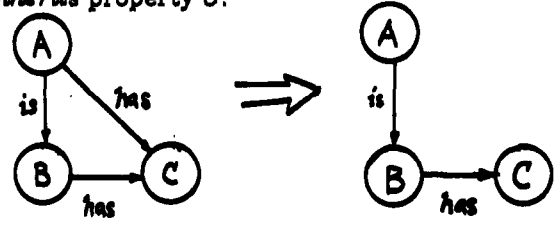
2.3. Connections, Abstraction, and 'Normalization'

Abstraction notions are important for modeling complex role arrangements among entities in the database. This is particularly true, for example, in modeling recursive relationships such as *PART*s having sub*PART*s, *EMPLOYEE*s having *MANAGER*s which are in turn *EMPLOYEE*s, etc. Recently there has been some formal work on these notions. Researchers have concentrated on generalization (ISA) hierarchies [SS], [BK], [Sc1], [HY], aggregation and FDs [SS], [HY], and connections [WM].

The examples in §2.1 suggest that abstraction notions are important in modeling of connections between objects in a database. Note that *is* and *has* information is easily obtained from Entity-Relationship diagrams, and embodies associations (facts) well, since it corresponds directly to natural language constructs. Moreover, the abstraction primitives *is* and *has* may be viewed as two important types of functions: inclusions, and many-to-one functions.

The *is* and *has* abstraction mechanisms also play important roles in design. Recall: The main emphasis of logical design is to eliminate redundant (or incorrect) connections from a schema, while preserving other connections and keeping integrity constraints easy to enforce. If all connections are modeled by functions, then redundancy is eliminated by removing functions which duplicate or are compositions of other functions in the database.

Thus, if all functions are either inclusions (*is*) or many-to-one relationships (*has*), then we must eliminate redundant compositions of these functions. We argue below that elimination of redundant connections is closely related to elimination of *property inheritance* (a term popular in semantic data models). If A is a subtype of B (A *is* B), and B has property C (B *has* C), then A *inherits* property C.



The connection between A and C is redundant here, and its elimination from the database yields an *enhanced* schema. This notion is of significance in schema transformation, and will be discussed at greater length in §5.

3. Motivation

The LAURA model was developed for the **logical design** *of* **large** *databases.* This statement is important, since many data models are developed to have a pleasant user interface; and assumptions made in existing theoretical results appear workable mainly in smaller databases [AP]. We argue in this section why formal data models like LAURA are needed. We encourage work along these lines; the recently developed Structural model of Wiederhold & El-Masri [WM] and FORMAT model of Hull & Yap [HY] are welcome additions.

In our view, it is undesirable to *begin* the Schema Design process with only notions of dependencies among attributes. Instead, it is desirable to use a model which can support abstractional concepts, entities and relationships adequately. Formal means may then used to *analyze* an initial rough schema and improve it through a sequence of *schema transformations* $T: S \to S'$ which take a schema S and yield an 'improved' schema S'. We call this process *Schema Enhancement*. (We avoid the word 'Normalization'.)

We expand here on the limitations of the relational model for representing real-world data. We restrict our attention to the relational model, since the bulk of formal research has dwelled on it, and since its limitations reflect the positive aspects of LAURA well. The Relational model is extremely powerful in its generality, and elegant in its mathematical foundation. It has provided many fruitful avenues of research over the past decade and a half. However, we can summarize three general problems with using the Relational model as a formal data model for the design of large databases: restrictive semantic capacity of the model itself, limitations of Functional Dependencies as a data modeling construct, and problems arising from a foundation on single-relation semantics.

3.1. Limited semantic capacity of the Relational model

Codd has underscored the need for additional semantic constructs in the Relational model for accuracy in real data modeling [C], as have other researchers [HM], [MS]. Efforts to improve the model by adding more constructs have concentrated on the spots where the model is weakest: adding the notion of entities, or objects, and incorporating constructs modeling connections, or associations, among objects. More work is needed to make dependencies properly convey entity concepts and existence semantics.

In addition, the bare Relational model presents difficulties in large database environments due to its putting extra semantic weight on attributes [U2]. The Relationship Uniqueness

Assumption [AP] forces all connections between a set of attributes to mean the same thing, although this is unnatural. The next section reviews some consequences of this assumption.

The bare Relational model has difficulty in expressing abstractional notions such as generalization and aggregation. Abstraction is essential in large databases, but the Relational model requires serious changes to accomodate multiple relationships between entities or roles of entities. Furthermore, the idea of 'property inheritance' introduced in §2.3 is lost when a schema is translated into a set of dependencies and relation schemes.

Finally, Kent points out in [K2] that the Relational model (and other record-oriented models) suffers many shortcomings in modeling certain types of data. It is particularly poor at modeling entities possessing a number of *categories* into which the entity can be classified, each having somewhat different properties. Kent gives clothing as an example: There are many types of clothing, and each type has individual features not shared by the others. Entities of this nature do *not* fit well in a uniform record structure, and are not treated effectively by the Relational model.

3.2. Limitations of Functional Dependencies

We begin by remarking that Functional Dependencies (FDs) alone are not sufficient for a formal notion of 'facts'. The incongruity between FDs and facts has been noted by by Smith and Smith [SS1], who argue that FDs do not correspond to facts ('aggregated objects') and that 'well-defined' relations should correspond to one natural language name rather than to a FD.

A related complaint is that FDs may represent either a constraint or a fact, or both, or neither. This is an elaboration of the observation in [FMU,p.347] that FDs can be of either a 'data structuring', or 'incidental', nature. Constraints are 'structuring' if they should be taken into account in developing a schema. However, some constraints are not; they should not influence the way the database should be laid out [LP].

Different ways of interpreting FDs is a major source of problems in database theory. Many attempts have been made to extend the elegant results holding with single relations to work for the multi-relation case. The Relationship Uniqueness Assumption (RUA) mentioned in §3.1 is one such attempt, but leads to counterintuitive results in large databases. From the set of FDs [AP]

$$EMP\# \rightarrow OFC\#,\ OFC\# \rightarrow PHONE$$
$$EMP\# \rightarrow LAB\#,\ LAB\# \rightarrow PHONE$$

one infers the FD $EMP\# \rightarrow PHONE$, implying surprisingly that an employee has a single phone.

Further problems arise if one assumes that all FDs are constraints that must be enforced on the database (the 'global consistency' problem [LTK], [Sa], [KS1,KS2]). In the example above, enforcing the global constraint $EMP\# \rightarrow PHONE$ is not guaranteed by simple enforcement of the four initial FDs. Naturally it is possible that we *would* want this FD to hold and be enforced, but this is not necessary.

Frequently the assumptions made concerning FDs are imprecise. In the literature it is common to find the belief that the FDs

$$X \rightarrow Y,\ Y \rightarrow X$$

imply a one-to-one map between the domains of X and Y. However, this inference is *not correct a priori*. It requires the RUA (in which case $Y \rightarrow X$ represents the inverse function of $X \rightarrow Y$), or the assumption that all FDs are total and onto.

3.3. Foundation on Single-Relation Semantics

The single-relation Relational model is quite elegant, so it is desirable for researchers to investigate its general applicability. Research into Join Dependencies (and Multivalued dependencies) has a number of goals, but certainly one of them is their reduction of multi-relation databases to a single relation (Universal Relation) foundation. Unfortunately, these dependencies have a circular definition [AP], and certainly do not mirror the semantics of large databases. It is not easy to fathom what Multivalued dependencies and 5th Normal Form actually *mean* in a large database.

Research on Universal Relation databases is still at an early stage [U2]. However, we see many practical reasons for avoiding a Join Dependency or Universal Relation outlook on the design of large databases. Real designers do not start out with a single relation. (And do not want to end up with one!) Second, using a single relation destroys the ability to model inclusions. Third, various Universal Relation assumptions force limitations mentioned in §3.1-3.2. Finally, the UR model introduces problems in the elimination of 'redundancy' in 'normalization' and the treatment of null values. The interested reader should consult [U2], [AP].

4. LAURA: The DBDG Model

In this section we introduce LAURA. LAURA is a data model expressly designed to capture functional associations precisely. It incorporates concepts from the Binary model and from Semantic Networks, as well as abstractional concepts in semantic models. In ways, the model is reminiscent of the Functional Models developed by Kerschberg and others ([KP,SK,HK], [BF], [HWY], [Sh]), but emphasizes the notion of connections (§4.3) and places weight on applications in *design* rather than in, say, query processing. LAURA's ties to Semantic Networks here are, to our knowledge, new.

After describing essential aspects of LAURA and how they are used in practice, we define *connections* and show some of the advantages of working with functional associations.

4.1. Formal Definition of LAURA

A LAURA database schema $\mathbf{S} = (\Omega, F)$ is a labelled hypergraph composed of:

(1) A set Ω of attributes defining the nodes of the graph. Attributes are of two *types*: *Extensional*, or *Intensional*. Each node or attribute A in Ω has a corresponding a *domain*, called $dom(A)$.

(2) A set F of connections, which are directed edges or hyperedges among attributes in Ω. Ordering of attributes in hyperedges is significant, in the sense that in $X \rightarrow Y$ the relative ordering of attributes in both X and Y is meaningful. Both X and Y should thus be viewed as *sequences* of attributes. Each edge/hyperedge is labelled with one of twelve functional connection types:

$-is\rightarrow$	$-\underline{is}\rightarrow$	$-\overline{is}\rightarrow$	$-\underline{\overline{is}}\rightarrow$
$-has\rightarrow$	$-\underline{has}\rightarrow$	$-\overline{has}\rightarrow$	$-\underline{\overline{has}}\rightarrow$
$-equiv\rightarrow$	$-\underline{equiv}\rightarrow$	$-\overline{equiv}\rightarrow$	$-\underline{\overline{equiv}}\rightarrow$

A LAURA schema also obeys the following *restrictions*:

R1 Intensional attributes have no outgoing edges.

R2 Edges or hyperedges may be labelled as $-\underline{is}\rightarrow$ only if they connect extensional attributes to intensional attributes.

R3 If $X \rightarrow Y$ is a hyperedge, then no proper (permuted) subsequence of X or Y may have outgoing edges to extensional attributes. However, X and Y may be involved in other hyperedges (or edges if X or Y is a single attribute).

R4 If $X - \sigma \rightarrow Y$ is a hyperedge, where σ is one of $-is\rightarrow$, $-\underline{is}\rightarrow$, $-\overline{is}\rightarrow$, or $-\underline{\overline{is}}\rightarrow$, then X and Y must be sequences of equal length.

R5 The set F must be both *onto-acyclic* and *total-acyclic*. (These terms will be defined below, in §4.3.)

Each edge or hyperedge $X - \sigma \rightarrow Y$ in F represents a function from $dom(X)$ to $dom(Y)$ to be stored in the database. (The function may be viewed alternatively as a constraint.) As mentioned above, there are twelve function types. We may describe a type σ by a four-tuple of 0-1 variables

$$\sigma = \langle Total, Onto ; Inclusion, One\text{-}to\text{-}one \rangle.$$

For example, <1,0;0,1> and <0,1;1,1> are function types. These variables make four statements about the nature of a function.

Total — If $Total = 1$, the function is a total function from $dom(X)$ to $dom(Y)$. In other words, for every element in $dom(X)$, the function associates a specific element in $dom(Y)$. This combines a functional constraint with an existence constraint, asserting that a Y exists for each X.

Onto — If $Onto = 1$, the function from $dom(X)$ to $dom(Y)$ is onto, so each element in $dom(Y)$ must have one or more elements in $dom(X)$ mapped onto it. This again combines functional and existence constraints.

Inclusion — If $Inclusion = 1$, then the function represents simply an identity map on $dom(X) \cap dom(Y)$. (This can be interpreted as saying that X and Y are joinable. Related notions appear in [K1].) Of necessity this situation requires the next variable, *One-to-one*, also to be 1.

One-to-one — If $One\text{-}to\text{-}one = 1$, the function must be a one-to-one map of $dom(X)$ into $dom(Y)$.

Thus, a function type <0,0;0,0> simply characterizes a (possibly many-to-one) function from $dom(X)$ to $dom(Y)$; and a function type <0,0;1,0> is not possible, since it implies a many-to-one inclusion relationship.

We use the following symbols for function type labels in graphical presentations of LAURA schemas: if σ is the function type

$$\sigma = \langle \alpha_1, \alpha_2; \alpha_3, \alpha_4 \rangle$$

then σ has the labeling indicated by the table below.

σ	$\alpha_3\alpha_4 = 11$	$\alpha_3\alpha_4 = 01$	$\alpha_3\alpha_4 = 00$
$\alpha_1\alpha_2 = 00$	$-is\rightarrow$	$-equiv\rightarrow$	$-has\rightarrow$
$\alpha_1\alpha_2 = 10$	$-\underline{is}\rightarrow$	$-\underline{equiv}\rightarrow$	$-\underline{has}\rightarrow$
$\alpha_1\alpha_2 = 01$	$-\overline{is}\rightarrow$	$-\overline{equiv}\rightarrow$	$-\overline{has}\rightarrow$
$\alpha_1\alpha_2 = 11$	$-\underline{\overline{is}}\rightarrow$	$-\underline{\overline{equiv}}\rightarrow$	$-\underline{\overline{has}}\rightarrow$

Thus '*is*' edges are inclusions, '*equiv*' edges are one-to-one maps, and '*has*' edges are many-to-one maps.

By restricting the domain and range of a function, we can make the function total or onto.

Definition A *Simple Qualified Attribute* $X.Y$ (resp. $Y{:}X$) in the context of a function $f : X \rightarrow Y$ is a symbol in Ω with an associated domain equal to precisely that part of $dom(Y)$ (resp. $dom(X)$) mapped onto by f.

In the notation of §2.2, $dom(X.Y) = dom(Y)_f$, and $dom(Y{:}X) = dom(X)_f$. Thus regardless of the type of function f, the restricted function $f : X \rightarrow X.Y$ is onto, $f : Y{:}X \rightarrow Y$ is total, and $f : Y{:}X \rightarrow X.Y$ is total and onto.

Definition A *database instance* of a valid LAURA schema $\mathbf{S} = (\Omega, F)$ is a set of tables $\boldsymbol{d} = \{\langle R_i, r_i \rangle \mid 1 \le i \le m\}$ where each R_i is a subset of Ω, and $r_i \subset dom(R_i)$. Specifically,

$$\{R_i \mid 1 \le i \le m\} = \{A \mid A \in \Omega, A \text{ is extensional}\} \cup \{X \cup Y \mid X - \sigma \rightarrow Y \in F\},$$

so all non-extensional attributes and all functions in F are embodied in the relation schemes of the database instance. Furthermore, the set $\{r_i\}$ obeys the functions in F. (Note: r_i never contains 'null values'.) Thus functional schemes $R_i = X \cup Y$ will have r_i obeying $X \rightarrow Y$, while unary schemes $R_i = A$ will have unary tables r_i containing all values of $dom(A)$ appearing in other functional schemes in the instance.

This definition requires storage of *all* function types, so even inclusions will be explicitly stored in tables. This is done only for the sake of simplicity.

As an example of a LAURA schema, in Figure 1 we see a portion of a Bill of Materials database. Attributes are circled, and edges are marked with their function types. The example exhibits the use of recursive relationships (assemblies and their component parts), weak entities (vendor locations), and potentially null fields (structure description, vendor status). These useful constructs are difficult to capture in some data models.

4.2. Practical Use of LAURA

In practice, LAURA is used in restricted ways which model common constructs such as entities, relationships, keys, foreign keys, etc. We have chosen to separate this usage from the formal definition of the model itself. Below we enumerate some common restrictions applied in modeling with LAURA.

(1) *Hyperedges are used only to represent relationships involving multiattribute keys and multiattribute foreign keys.* We find it helpful to equate (using an $-equiv\rightarrow$ edge) a set of attributes with a single 'surrogate' attribute, which then participates in other relationships. This tends to reduce the complexity of a graphical schema display. Conceivably hyperedges could also be used in other abstractional ways, through aggregation of large sets of attributes.

Each entity in Figure 1 is represented with an external key/surrogate attribute, displayed in capital letters. Thus the multiattribute key *COMPONENT:PART#,ASSEMBLY:PART#* is *equivalent* to the external key *STRUCTURE*.

(2) *All inclusion edges are required to be* total. Furthermore, LAURA requires the *is* and *has* edges to comprise respective *ISA* and '*HASA*' hierarchies (directed acyclic graphs), much as in [SS1]. Generalization and abstraction concepts are captured in this manner. It is rare that these hierarchies are more than a few levels deep. The *ISA* hierarchy can have perhaps 3 or 4 levels (Fahlman [Fa,p.39] concurs, noting that even the entire hierarchy of living creatures has only 20 levels), while in our experience *HASA* is very short, with 2 or 3 levels at most. Note that in LAURA, intensional attributes represent 'datatypes', while extensional attributes are always stored subsets of the corresponding type.

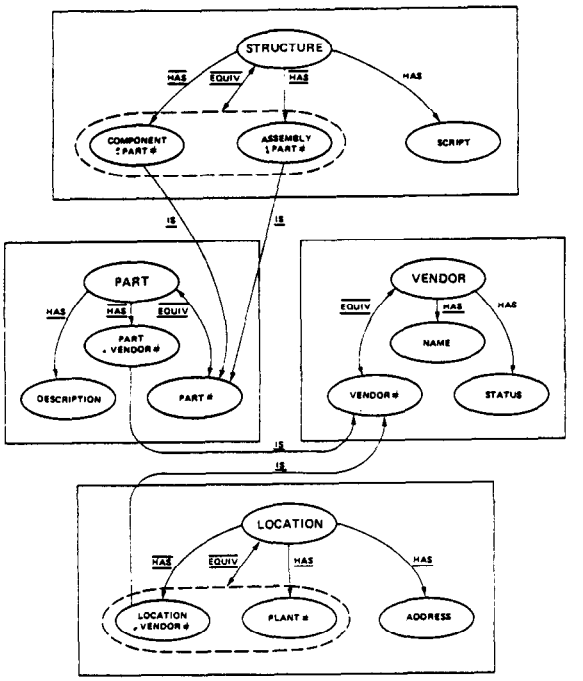
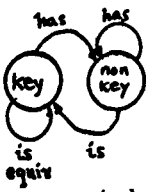


Figure 1. LAURA Schema for Bill of Materials Database

(3) *Many different attribute types are introduced.* We have found it fruitful to distinguish between entities, keys, parts of keys, foreign keys, category discriminators, properties, and so forth. This permits enforcing restrictions on the types of functions that can connect one type of attribute with another. For example, an $-is\rightarrow$ relationship can connect only two key attributes. The table and diagram below illustrate the restrictions possible between individual attributes or sets of attributes. Existence constraints could of course be added to these restrictions.

key	$-is\rightarrow$	key
key	$-equiv\rightarrow$	key
key	$-has\rightarrow$	non-key
non-key	$-has\rightarrow$	non-key
non-key	$-is\rightarrow$	key



(4) *Many-many relationships are represented by creating a surrogate attribute (external key) and using a multiattribute $-equiv\rightarrow$ connection.* Similar methods can be used for n-ary relationships. Other data models provide more flexibility in modeling such relationships, but 'user-friendliness' is not the emphasis here.

(5) *Possession and roles are expressed with simple qualified attributes.* The attributes *entityname.object*, *rolename:entityname* such as *PART.VENDOR#*, and *ASSEMBLY:PART#* in Figure 1 serve to incorporate this structuring. Distinct roles of an entity are thus assigned a unique attribute in Ω.

4.3. Connections in LAURA

In this section we show how connections may be defined using the *is* and *has* connections of §4.1. Generally speaking, a connection in a database might be the result of any computable query (mapping) on the database. This would not lead to a useful working definition, unfortunately. We restrict our attention here to connections defined by project-join mappings, and concentrate on connections which define functions.

More formally, let $\mathbf{S} = (\Omega, F)$ be a valid LAURA schema, with the implied relational schema $\{R_i\}$ for its database instances.

Definition Let S, T be sequences of attributes from Ω. A *connection* $C: S \rightarrow T$ is a finite project-join map on the relational schema,

$$C = \pi_{S,T}((\cdots(R_{i_0} \underset{X_1=X_1}{|\times|} R_{i_1}) \underset{X_2=X_2}{|\times|} \cdots) \underset{X_n=X_n}{|\times|} R_{i_n})$$

defining a relationship between $dom(S)$ and $dom(T)$. Here each R_i is at least binary, and for all $j>0$ X_j is a sequence of attributes included in both R_{i_j}, and in the attributes of $R_{i_0}, \ldots, R_{i_{j-1}}$.

It may be helpful to view a connection as a hypergraph on attributes, where relation schemes are represented as hyperedges, and binary edges connect specific attributes to indicate equijoin of the corresponding columns. The notion of connections here is related in ways to the earlier work of Lien, Sagiv, and Kuck & Sagiv [L1,L2,L3], [Sa], [KS1,KS2], but differs substantially in that connections are not restricted to a traversals of the schema (relation schemes R_i can be used multiple times, or not at all).

We are particularly concerned with functional connections, since they have interesting inference and redundancy properties.

Definition Let C be a connection for a schema $\mathbf{S} = (\Omega, F)$. We say C is a *functional connection* $C: S \rightarrow T$ if it defines a function from $dom(S)$ to $dom(T)$ for all database instances of the schema. In this case we say $F \models C$, or F *logically implies* $C: S \rightarrow T$.

Recall as in §2 that connections can be obtained as chains of functional associations. For example,

(1) *MGR* $-\underline{is}\rightarrow$ *EMP* $-\underline{is}\rightarrow$ *SOC_SEC_DONOR*
(2) *MGR* $-\underline{is}\rightarrow$ *EMP* $-\underline{has}\rightarrow$ *MGR* $-\underline{is}\rightarrow$ *EMP* $-\underline{has}\rightarrow$ *SAL*
(3) *DELIVERY_VAN#* $-\underline{is}\rightarrow$ *TRUCK* $-\underline{is}\rightarrow$ *VEHICLE*
(4) *PROJECT* $-\underline{has}\rightarrow$ *MGR* $-\underline{is}\rightarrow$ *EMP*

all give functional connections. Quick study of these examples makes us realize that the following connections must hold in each case:

(1) *MGR* $-is\rightarrow$ *SOC_SEC_DONOR*
(2) *MGR* $-has\rightarrow$ *SALARY*
(3) *DELIVERY_VAN#* $-\underline{is}\rightarrow$ *VEHICLE*
(4) *PROJECT* $-\underline{has}\rightarrow$ *EMP*

In other words, a composition of functions is total if all of the functions involved are total, onto if they are all onto, and an inclusion if all the functions are inclusions.

We have found it convenient to introduce a notation for chains such as the ones here.

Definition A *Qualified Attribute* $X_1.X_2.\cdots.X_n$ in the context of a schema (Ω, F) is a sequence of attributes such that

$$X_0 => X_1 => \cdots => X_n$$

is a connection, where "=>" is some function in F. For example,

PROJECT.MGR.EMP.OFFICE.PHONE
EMP.MGR.EMP.MGR.EMP.SALARY

are qualified attributes. From the attribute set Ω we can construct a *complete qualified attribute set* consisting of Ω and all qualified attributes constructible using F.

More generally, if S is a sequence of attributes, we would like to find other attributes connected (related functionally) to S. The question the reader must be asking himself is *how* one can find functional connections logically implied F. We can construct a set of rules which give sound inferences concerning the implication of connections.

Below let σ, τ, ρ be function types as in §4.1, where $\sigma = <\alpha_1,\alpha_2;\alpha_3,\alpha_4>$, and $\tau = <\beta_1,\beta_2;\beta_3,\beta_4>$.

Definition *reverse* (σ) is a function type which is defined only if σ is one-to-one ($\alpha_4=1$), in which case

$$reverse(\sigma) = <\alpha_2,\alpha_1;\alpha_3,\alpha_4>.$$

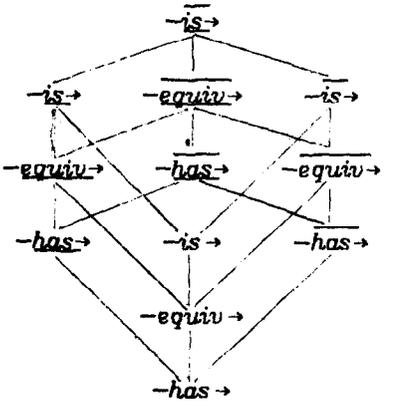
For example, *reverse* $(-\underline{is}\rightarrow) = -\overline{\underline{is}}\rightarrow$, and *reverse* $(-\underline{equiv}\rightarrow) = -\underline{equiv}\rightarrow$.

Definition *compose* (σ,τ) is a function type, defined by

$$compose(\sigma,\tau) = <min(\alpha_1,\beta_1), min(\alpha_2,\beta_2); min(\alpha_3,\beta_3), min(\alpha_4,\beta_4)>.$$

For example, *compose* $(-\underline{is}\rightarrow, -\overline{\underline{has}}\rightarrow) = -has\rightarrow$, and *compose* $(-\underline{equiv}\rightarrow, -\underline{is}\rightarrow) = -\underline{equiv}\rightarrow$.

Definition *implies* (σ,τ) is a predicate which is true if a function of type σ is also of type τ. In general, a function type $\sigma = <\alpha_1,\alpha_2;\alpha_3,\alpha_4>$ implies all functions types $\tau = <\beta_1,\beta_2;\beta_3,\beta_4>$. where $\beta_i \leq \alpha_i$, $1 \leq i \leq 4$ (or, equivalently, $\beta_i => \alpha_i$, $1 \leq i \leq 4$, if we regard 0-1 variables as Boolean variables). *implies* defines a lattice on function types:



With these definitions, we obtain the following rules for inferring functions $S \rightarrow T$ from a set of connections F. Below let X,Y,Z denote arbitrary sequences of attributes.

I0. (Reflexivity)
$X -\underline{is}\rightarrow X$.

I1. (Permutation)
if $X -\sigma\rightarrow Y$, **then** $\pi_1(X) -\sigma\rightarrow \pi_2(Y)$, where π_1,π_2 are permutations. When σ is an inclusion, we require $\pi_1=\pi_2$.

I2. (has-Projection)
if $X -\sigma\rightarrow Y$, **then** $X -\sigma\rightarrow Y'$, where σ is many-to-one (*has*), and Y' is an arbitrary subsequence of Y.

I3. (is-Projection)
if $X -\sigma\rightarrow Y$, **then** $X' -\sigma\rightarrow Y'$, where σ is an inclusion (*is*), and X' and Y' are corresponding subsequences of X and Y.

I4. (Type Implication)
if $X -\sigma\rightarrow Y$, **then** $X -\tau\rightarrow Y$, where *implies* (σ,τ).

I5. (Reversal)
if $X -\sigma\rightarrow Y$, **and** *reverse* (σ) is defined, **then** $Y -\tau\rightarrow X$, where τ=*reverse* (σ).

I6. (Augmentation)
if $X -\sigma\rightarrow Y$, **then** $XZ -\sigma\rightarrow YZ$.

I7. (Composition)
if $X -\sigma\rightarrow Y$, **and** $Y -\tau\rightarrow Z$, **then** $X -\rho\rightarrow Z$, where ρ = *compose* (σ,τ).

Definition Let $C: S -\sigma\rightarrow T$ be a functional connection. We say that C *is implied by* F *using the rules*, or $F \mid\!- C$, if there is some derivation of $S -\sigma\rightarrow T$ using the functions of F represented by schemes in C.

It is clear that the rules above are sound. It would be pleasant if we could prove they are also complete for deriving functional connections in general, i.e., prove that

$$F \mid\!= C \text{ iff } F \mid\!- C.$$

Unfortunately we cannot: there are many ways to do equijoins among functions, and rule I7 only gives one.

For example, suppose we are given two functions $A -\underline{has}\rightarrow B$ and $B -\underline{has}\rightarrow A$. The connection which "joins" these two functions on both A and B will satisfy $A -equiv\rightarrow B$. None of the rules above covers this situation.

Unfortunately there are quite a number of ways in which one can join two functions, since their attribute sequences can intersect in 16 different ways and there are 12^2 possible function type combinations. Admittedly, this space of alternatives is not that vast, but we have not attempted to derive these rules. However, we

Conjecture If the rule set I0-I7 above is augmented to include all rules detailing inferences following from the join of two functions, then

$$F \mid\!= C \text{ iff } F \mid\!- C.$$

We sketch a possible proof line below, once some important properties of LAURA schemas are explained.

Definition A schema $\mathbf{S} = (\Omega,F)$ is *onto-acyclic* (resp. *total-acyclic)* if there is no sequence of distinct edges

$$X_0 \rightarrow Y_0, X_1 \rightarrow Y_1, \ldots, X_n \rightarrow Y_n$$

with

$$Y_i \cap X_{i+1 \bmod n+1} \neq \phi,\ 0 \leq i \leq n,$$

which all have onto (resp. total) function types. Here each edge $X_i \rightarrow Y_i$ either in F, or its reversal is in F, but no edge can occur more than once is the cycle (in original or in reversed form).

These acyclicity notions are important since they let us complete partial counterexample databases to full counterexample databases, avoiding the strange problems of [CFP]. Let a *partial instance* d of a (valid) LAURA schema (Ω,F) be a database instance which does not violate any function in F. We say d is a *complete instance* if it also satisfies F.

Lemma Any (finite) partial instance of a LAURA schema can be extended to a (finite) complete instance.

The extension constructed to prove this lemma is essentially that of the 'chase' technique used in the literature (e.g., [JK]) of repeatedly taking an unsatisfied constraint and adding new things to the database instance to help satisfy it, but the trick here is that the acyclicity required of LAURA schemas limits the extent of this extension. When the partial instance is finite, the chase process terminates after a finite number of steps, since the onto- and total-acyclicity guarantee that no symbol added to some table r_j in a database instance can indirectly cause another symbol to be added to r_j later on. We omit the details. ▪

The acyclicity conditions avoid problems noticed by [CFP]. Consider the example with two relations $R_0(AB)$, $R_1(CD)$ FDs $A \rightarrow B$, $C \rightarrow D$, and INDs $R_0[A] \subseteq R_1[D]$, $R_1[C] \subseteq R_0[B]$ as in [CFP]. These dependencies are argued to imply the inequalities

$$card(A) \geq card(B) \geq card(C) \geq card(D) \geq card(A),$$

where $card(A)$ gives the number of distinct entries in the A-column, etc., because the FDs give many-one constraints and inclusions also give such inequalities. From these it follows that all cardinalities are equal, whence $R_0[B] \subseteq R_1[C]$ and $R_1[D] \subseteq R_0[A]$. This conclusion does not follow ordinary inference rules. However, this inference relies on four significant assumptions:

(1) *All relations are finite.*
(2) *All INDs represent total functions.*
(3) *All FDs represent total functions.*
(4) *All FDs represent onto functions.*

These assumptions imply an onto-cycle. The last two assumptions require that the database have no null values where FDs exist, and are strong by practical standards.

A possible proof line for the conjecture follows by induction on n, the size of the connection $C: S-\sigma\rightarrow T$.

Basis
When $n=0$ the connection has the form

$$C = \pi_{S,T}(R_{i_0})$$

and it is possible to show that the rules I1-I6 are all that is needed. There are several cases, depending on R_{i_0} and σ, but in each case one can construct a table r_{i_0} which contradicts any $S-\sigma\rightarrow T$ that does not follow from the rules. This table can be extended to a complete database instance because of the fact that valid LAURA schemes are onto-acyclic and total-acyclic (§4.1, restriction R5).

Induction Step
(This step in the proof is not yet complete.) It is sufficient to show that if a function $S-\sigma\rightarrow T$ holds in a connection, then it results from either (1) a function on the connection over $R_{i_0}, \ldots, R_{i_{n-1}}$, (2) a function on R_{i_n}, or (3) the join of these two functions. Many cases arise again. Interestingly, this step will not follow without LAURA restrictions R1-R3, because of 'pullback functions' [M]. For example, if we join the functions $AB-\underline{is}\rightarrow CD$ and $C-has\rightarrow D$, we obtain a connection which satisfies $A-is\rightarrow B$, the 'pullback' of $C-has\rightarrow D$. ▪

5. Schema Enhancement with LAURA

One motivation for investigating functional data models as we have here is that doing so provides a formal notion of redundancy. Generally speaking, we would like to say a database schema is *redundant* if there are two different computable queries (mappings, connections) which yield the same result for every database instance of the schema. Unfortunately this notion of redundancy is very difficult to make precise. We content ourselves with redundancy defined by functional connections obtained through composition.

The designer's job at this point is to remove as much error and redundancy as possible from the schema, while simultaneously making sure that integrity constraints are not difficult to enforce. (If necessary, redundancy can always be reintroduced later to improve query processing performance or ease of constraint enforcement.) Unfortunately, there are many ways that redundancy can creep into a schema, and there is no best way to detect redundancy automatically or to eliminate it. A sound approach here is to equip the designer with tools for improving the schema, and permit him to proceed thusly armed on his own.

5.1. Schema Transformations

The schema enhancement process we use can be thought of as beginning with a LAURA graph and proceeding through a sequence of *transformations*. Edges or attributes from the graph are added or deleted so as to either correct errors in the graph, or eliminate redundancy (such as caused by equivalent connections). Transformations used by the Database Design Group consist of adding or deleting an edge and/or a attribute, and relabelling or reconnecting edges and attributes as necessary. We may group the transformations loosely into two classes:

(1) *Local Enhancement*
Transformations that make minor additions or deletions, producing a schema which is more accurate (though formally inequivalent) to its predecessor; and

(2) *Redundant Connection Elimination*
Transformations that delete derivable connections, producing an equivalent schema.

The second class is more interesting mathematically, but unfortunately the first class is very necessary. There is always ambiguity and error in a design after the view merging phase, and these flaws should not be propagated further in the design.

These modifications may be made in such a way as to minimize some quantity, such as the number of remaining equivalent connections in the graph (to make constraint enforcement easy), or the number of edges in the graph (to minimize the number of stored associations).

We feel significant benefits come from operating on a LAURA schema rather than on a relational schema in the schema enhancement process. Critical information such as existence constraints or entity constructs is either lost or represented clumsily when stored using relations and dependencies. We will show below how much of normalization amounts to a desire to eliminate the storage of inherited properties. This understanding is obscured (for the authors at least) when expressed in the formalism of FDs.

In the remainder of this section we give a brief presentation of some redundant connection elimination methods. Although our explanation is simplified, we wish to emphasize that determining when transforms are to be applied is not simple.

5.2. Redundant Connection Detection & Elimination

Two general problems must be dealt with here:

(1) Finding redundancy in the schema.

(2) Eliminating as much of it as possible.

Redundancy is never implied by a schema *a priori*. Semantic information is required to determine equivalence of connections.

Let us define redundancy in the following way:

Definition Let (Ω,F) be a LAURA schema, and $C_1, C_2: S \rightarrow T$ be connections obtained from F by rules I0-I7. We say C_1 is ***redundant*** if there exists a syntactically inequivalent connection C_2 such that C_1 and C_2 define identical functions for all database instances of the schema.

The inference rules of §4 may be used to develop an interactive mechanism for finding redundancy. A basic approach is to make various ***assumptions*** about which connections are redundant, and use these assumptions to search for redundancy. These assumptions state that if connections are ***syntactically equivalent*** (under some definition of syntactic equivalence) then the connections are assumed to be ***semantically equivalent*** as well. As equivalent connections are found they are announced and, unless the designer somehow denies the equivalence, they are marked to be dealt with.

It is desirable to make the assumptions explicit. There are many possibilities, including:

(1) Qualified attributes
If a qualified attribute is found which is type-equivalent to an edge in F, the edge is marked as redundant.

(2) Triangles
If any connection inferrable from F is found which is syntactically equivalent to an edge in F, the edge is marked as redundant.

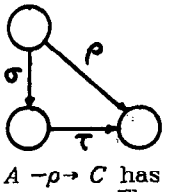
(3) Simple RUA
All connections $S \rightarrow T$ of the same basic type (*is*, *has*, *or equiv* – ignoring whether they are total or onto) are assumed equivalent, and must be dealt with somehow. If two connections are Simple RUA-equivalent but not fully type-equivalent, it is possible that some existence constraints have been forgotten in the schema.

(4) Deluxe RUA
All connections $S \rightarrow T$, ignoring type, are assumed equivalent.

These assumptions may be used in conjunction; the DBDG uses a set of heuristic assumptions for detecting redundancy.

Redundancy is commonly found in 'triangles'. Suppose that three attributes A,B,C have connections $A \text{ –}\sigma\rightarrow B$, $B \text{ –}\tau\rightarrow C$, $A \text{ –}\rho\rightarrow C$ such that the composition of the first two always equals the third. Thus $A \text{ –}\rho\rightarrow C$ is redundant, and $\rho = compose(\sigma,\tau)$. Consider briefly what can happen when σ and τ are either *–is →* or *–has →*. If we inspect the table and diagram

σ	τ	ρ
–has →	*–has →*	*–has →*
–has →	*–is →*	*–has →*
–is →	*–has →*	*–has →*
–is →	*–is →*	*–is →*



we find that eliminating the edge $A \text{ –}\rho\rightarrow C$ has different meaning in each of the four cases. The second case is prohibited under normal LAURA use. It appears that the only cases where eliminating the edge $A \text{ –}\rho\rightarrow C$ produces a difference in the corresponding relational schema is when σ = *–has →* and τ = *–has →*, or σ = *–is →* and τ = *–has →*. In the latter, more prevalent, case, C is an ***inherited property*** of A as in §2.3, and $A \text{ –}\rho\rightarrow C$ is a ***pullback function*** as in §4.3.

Much of the intent of normalization seems to lie in elimination of stored inherited attributes. This concept came as a surprise to the authors.

Generally speaking, however, more forms of redundancy must be dealt with than just 'triangles'. Abstractly the redundancy detection process should provide us with an ***equivalence relation*** on connections derived from F. The redundancy elimination process must make decisions on how to remove as much redundancy as possible. The DBDG again uses a heuristic process, but many interesting theoretical problems lie open in this area. For example, if connections are restricted to be paths in the schema graph, we arrive at a *Generalized Transitive Reduction Problem:*

> Given a (hyper)graph (Ω,F) and an equivalence relation between edges and paths of the graph, find a subgraph (Ω,F') with as few of the equivalence relation edges as possible, without losing any accessibility. Cf. [AGU].

Other similar problems are easy to formulate.

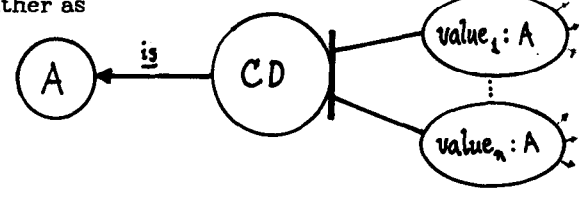
6. Extensions to LAURA and Future Directions

This paper has tried to convey the basic aspects of LAURA. We illustrate here additional avenues along which LAURA may be pushed, concentrating especially on the ***categorization*** primitives added to LAURA, which are missing in some data models. We also list a number of interesting problems open for further investigation.

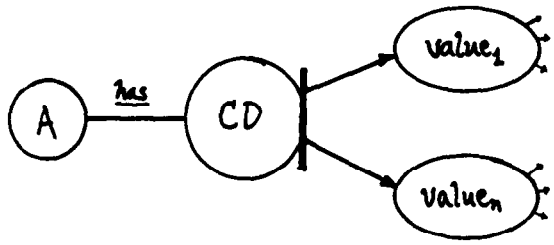
6.1. Categories and Disjoint Disjunctions

We mentioned in §3.1 Kent's argument that the Relational model (and other record-oriented models) is inadequate in modeling entities having a number of ***categories*** into which the entity can be classified, each having somewhat different properties. There are several possible ways to store such entities in a relational database, including (1) Make the relation scheme contain ***all*** attributes of all categories; (2) Use 'code fields', whose values define what the other fields in a record mean (The PASCAL variant record); (3) Decompose the relation 'horizontally' into disjoint subrelations, where each resulting relation corresponds to one of the original categories [FK], [F], [BP]; (4) Use unnormalized (non-1NF) relations.

In LAURA diagrams, category relationships are indicated using a specially labelled, directed hyperedge. These hyperedges may be drawn either as



for *–is→* or *–equiv→* relationships, or as



for *–has→* relationships. In both these situations, *CATEGORY DISCRIMINATOR* (*CD*) is a property of *A*, and $value_1, value_2, \ldots, value_n$ give a (not necessarily exhaustive) list of the possible values of the *CD*. The values of *A* are partitioned into different sets according to the value of *A*'s *CD*. The existence semantics are quite strong: the *CD* must always be defined.

The two categorization constructs are equivalent, and one may be transformed to the other. Property inheritance transforms also are commonly applicable here. Essentially the *–is→*/*–equiv→* construct corresponds to 'horizontal decomposition', alternative (3) at the beginning of this section, while the *–has→* construct corresponds to one of alternatives (1) and (2).

Unfortunately the notion of connections in §4 may have to be modified to incorporate categorization. Difficulties arise in determining exactly which path one can take without knowing the value of the *CATEGORY DISCRIMINATOR*. Somehow one must incorporate data from the database into the definition of connections (incorporate data into metadata). This situation is common in semantic networks, where schema information is stored together with the actual data [B],[Fa].

LAURA requires the category partitions to be *disjoint*. Requiring disjointness is not essential, and in fact complicates the model, since *–is→* edges between categories can yield invalid models. Currently we place a new type of edges, labeled

$$-isn't\rightarrow$$

between attributes whose domains must be disjoint, and extend the definition of *compose* in §4.3 so that

$$compose(-is\rightarrow, -isn't\rightarrow) =$$

$$compose(-isn't\rightarrow, -is\rightarrow) = -isn't\rightarrow.$$

The schema is taken to be valid as long as it is not possible to find an attribute *A* such that $A-isn't\rightarrow A$.

6.2. Directions for Future Research

The results presented here leave many directions open for further investigation. Many practical and theoretical problems spring to mind.

- Inference rules and their Completeness
  Develop inference rules for 'joins' of functions as in §4.3, proving them complete if possible. Is the Completness Conjecture of this section true?
- Categorization
  How can the definition of connections be extended with categorization primitives of §6.1? What sort of inference rules can be developed?
- Generalized Transitive Reduction
  Are there fast algorithms for solving the reduction problem in §5.2, for eliminating as many edges as possible from a graph without disturbing graph accessibility?
- Qualified Attributes
  The qualified attributes of §4.3 appear to have many nice properties, both from a practical and theoretical standpoint. It might be interesting to develop an algebra of qualified attributes. In addition, note that Mitchell's pullback function $p(U,V,B)$ [M] is equivalent to the qualified attribute $U.V.B$ (under the RUA). It seems possible that this close relationship between qualified attributes and Mitchell's existentially quantified attributes can be exploited, possibly through modification of Mitchell's rule FI3 (Attribute Introduction), to lead to an algorithm for (infinite) implication on sets of FDs and INDs.
- Acyclicity
  Find fast algorithms for determining whether a LAURA schema is onto-acyclic or total-acyclic, as defined in §4.3.
- Other function types
  How can LAURA be extended to include set-valued functions as in many functional models? This would permit a natural means of modeling many-to-many relationships, and would have interesting inference ramifications, because the *reverse* of many-to-one functions can be defined. Note [HK] also permits list-valued functions. It may also be interesting to extend LAURA to have edges labeled by function names (role names) and permit multiple edges between a pair of attributes. How do the *–isn't→* connections of §6.1 interact with other function types? Inference rules?

7. Conclusions

We have shown that LAURA has a number of desirable features as a formal data model:

Simple functional foundation
Powerful abstraction primitives
Categorization primitives

These features are advantageous when compared with other formal data models.

We feel LAURA exhibits significant advantages as a formal model, both for modeling of large real-world schemas, and for the simple, formally founded schema transformation process it encourages. LAURA's functional and categorization primitives work well with schema enhancement, and have exposed here the importance of eliminating inherited properties in improving a schema.

We have encountered a number of interesting problems in developing this model, both practical and theoretical. We encourage the development of new practical formal models, and a increased application in design practice of design theory.

Acknowledgements

The authors are indebted to Rick Hull for his meticulous study of an earlier draft of this paper; his comments and suggestions improved the paper considerably. The authors would also like to thank Kamran Parsaye for his simplification of some of the arguments here, and Dennis McLeod for his comments about the model. Members of the DBDG family have provided a great deal of helpful feedback and encouragement which contributed to the completion of this work. Comments by the referees also led to a number of improvements.

References

[A] Abrial, J., "Data Semantics", in Klimbie & Koffeman, eds., *Data Base Management*, North-Holland, 1974.

[AGU] Aho, A.V., M.R. Garey, J.D. Ullman, 'The Transitive Reduction of a Directed Graph', *SIAM J. Comput.* 1:2, 131-137.

[AP] Atzeni, P., D.S. Parker, 'Assumptions in Relational Database Theory', *Proc. First ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Los Angeles, CA, March 1982.

[BK] Beeri, C., & H.F. Korth, 'Compatible Attributes in a Universal Relation', *Proc. First ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Los Angeles, CA, March 1982.

[BG] Bernstein, P., N. Goodman, 'What does Boyce-Codd Normal Form Do?', *Proc. Sixth Intnl. Conf. on Very Large Data Bases*, Montreal, Canada, October 1980.

[BP] De Bra, P., & J. Paredaens, 'Horizontal decompositions for handling exceptions to functional dependencies', *Proc. Workshop on Logical Bases for Data Bases*, Toulouse, France, 14-17 December 1982.

[B] Brachman, R., 'On the Epistemological Status of Semantic Networks', in Findler, V. (ed.), *Associative Networks* NY: Academic Press, 1979.

[BF] Buneman, P., & R.E. Frankel, 'FQL – A Functional Query Language', *Proc. ACM SIGMOD 1979 Intnl. Conf. on Management of Data*, Boston, MA, May-June 1979, 52-58.

[CFP] Casanova, M.A., R. Fagin, & C. Papadimitriou, 'Inclusion Dependencies and their Interaction with Functional Dependencies', (Extended abstract) *Proc. First ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Los Angeles, CA, March 1982. Full paper: IBM Research Report RJ3380, March 1982.

[C] Codd, E.F., 'Extending the Data Base Relational Model to Capture More Meaning', *ACM Trans. Database Systems 4:4, December 1979, 397-434.*

[D] Date, C., 'Referential Integrity', *Proc. Seventh Intnl. Conf. on Very Large Data Bases*, Cannes, France, September 1981.

[F] Fagin, R., 'A Normal Form for Relational Databases That is Based on Domains and Keys', *ACM Trans. Database Systems 6:3*, September 1981, 387-415.

[FMU] Fagin, R., A.O. Mendelzon, J.D. Ullman, 'A Simplified Universal Relation Assumption and its Properties', *ACM Trans. Database Systems 7:3*, September 1982, 343-360.

[Fa] Fahlman, S., *NETL: A System for Representing and Using Real-World Knowledge*, Boston: MIT Press, 1979.

[FK] Furtado, A.L., & L. Kerschberg, 'An Algebra of Quotient Relations', *Proc. ACM SIGMOD 1978 Intnl. Conf. on Management of Data*, pp. 1-8.

[HM] Hammer, M., & D. McLeod, 'Database Description with SDM: A Semantic Database Model', *ACM Trans. Database Systems 6:3*, September 1981, 351-386.

[HK] Hecht, M.Y., & L. Kerschberg, 'Update Semantics for the Functional Data Model', Database Research Report No. 4, Bell Laboratories, Holmdel, NJ 07733, January 1981.

[HWY] Housel, B.C., V. Waddle, S.B. Yao, 'The Functional Dependency Model for Logical Database Design', *Proc. Fifth Intnl. Conf. on Very Large Data Bases*, Rio de Janeiro, Brazil, October 1979.

[HY] Hull, R., & C. Yap, 'The Format Model: A Theory of Database Organization', *Proc. First ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Los Angeles, CA, March 1982.

[JK] Johnson, D.S., & A. Klug, 'Testing Containment of Conjunctive Queries Under Functional and Inclusion Dependencies', *Proc. First ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Los Angeles, CA, March 1982.

[K1] Kent, W., 'The Entity Join', *Proc. Fifth Intnl. Conf. on Very Large Data Bases*, Rio de Janeiro, Brazil, October 1979, pp. 232-238.

[K2] Kent, W., 'Limitations of Record-Based Information Models', *ACM Trans. Database Systems 4:1*, March 1979, pp. 107-131.

[KP] Kerschberg, L., & J.E.S. Pacheco, 'A Functional Data Base Model', Tech Report, Pontifica Universidade Catolica do Rio de Janeiro, Brazil, February 1976.

[KS1] Kuck, S., & Y. Sagiv, 'A Universal Relation Database System Implemented via the Network Model', *Proc. First ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Los Angeles, CA, March 1982.

[KS2] Kuck, S., & Y. Sagiv, 'Designing Globally Consistent Network Schemas', *Proc. ACM SIGMOD 1983 Intnl. Conf. on Management of Data*, San Jose, CA.

[LP] LeDoux, C.H., & D.S. Parker, 'Reflections on Boyce-Codd Normal Form', *Proc. Eighth Intnl. Conf. on Very Large Data Bases*, Mexico City, Mexico, September 1982.

[L1] Lien, Y.E., 'On the Semantics of the Entity-Relationship Model', in *Entity-Relationship Approach to Systems Analysis and Design* (P.P-S. Chen, ed.), pp. 155-167, North-Holland, Amsterdam, 1980.

[L2] Lien, Y.E., 'Hierarchical Schemata for Relational Databases', *ACM Trans. Database Systems 6:1*, March 1981, 48-69.

[L3] Lien, Y.E., 'On the Equivalence of Database Models', *Journal of the ACM 29:2*, April 1982, 333-362.

[LTK] Ling, T.-W., F.W. Tompa, & T. Kameda, 'An Improved Third Normal Form for Relational Databases', *ACM Trans. Database Systems 6:2*, June 1981, 329-346.

[MS] McLeod, D., & J.M. Smith, 'Abstraction in Databases', *Proc. of the Workshop on Data Abstraction, Databases, and Conceptual Modeling*, Pingree Park, CO, June 23-26, 1980.

[M] Mitchell, J.C., 'Inference Rules for Functional and Inclusion Dependencies', *Proc. ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Atlanta, GA, March 1983.

[Sa] Sagiv, Y., 'A Characterization of Globally Consistent Databases and their Correct Access Paths', preprint, C.S. Dept., University of Illinois, 1981.

[Sc1] Sciore, E., 'Improving Semantic Specification in a Relational Database', *Proc. ACM SIGMOD 1979 Intnl. Conf. on Management of Data*, Boston, MA, May 30-June 1, 1979.

[Sc2] Sciore, E., 'Inclusion Dependencies and the Universal Instance', *Proc. Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Atlanta, GA, March 1983.

[Sh] Shipman, D.W., 'The Functional Data Model and the Data Language DAPLEX', *ACM Trans. Database Systems 6:1*, March 1981, 140-173.

[SK] Sibley, E.H., & L. Kerschberg, 'Data Architecture and Data Model Considerations', *Proc AFIPS NCC, Dallas, TX, June 1977, 85-96.*

[SS1] Smith, J.M., D.C.P. Smith, 'Database Abstractions: Aggregation', *Communications of the ACM 20:6*, June 1977.

[SS2] Smith, J.M., D.C.P. Smith, 'Database Abstractions: Aggregation and Generalization', *ACM Trans. Database Systems 2:2*, June 1977, 105-133.

[TL] Tsichritzis, D.C., & F.H. Lochovsky, *Data Models*, NY: Prentice-Hall, 1982.

[U1] Ullman, J.D., *Principles of Database Systems*, Computer Science Press, Potomac, MD, 1980.

[U2] Ullman, J.D., 'The U.R. Strikes Back', *Proc. First ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Los Angeles, CA, March 1982.

[WM] Wiederhold, G., & R. El-Masri, 'A Structural Model for Database Systems', Technical Report STAN-CS-79-722, Computer Science Dept., Stanford University, February 1979.