LOGICAL INTERPRETATION OF RELATIONAL DATABASES

Ronald M. Lee

December 1981
WP-81-163

INTERNATIONAL INSTITUTE FOR APPLIED SYSTEMS ANALYSIS
2361 Laxenburg, Austria

ABSTRACT

The reformulation of data management type databases in a formal, logical calculus is described. Advantages of this logical form are to provide a framework for automatic inferencing on the database as well as a formal clarification of the databases semantics. Principle applications are to artificially intelligent managerial decision support systems.

CONTENTS

INTRODUCTION	1
DATABASE INFEENCING	2
SEMANTIC ISSUES	3
OUTLINE OF PAPER	4
REVIEW OF THE RELATIONAL MODEL	4
PREDICATE CALCULUS—REVIEW AND NOTATION	6
A MULTI-SORTED LOGIC	8
NOTE ON CONTROL IMPLICATIONS	10
TRANSLATION FROM RELATIONAL MODEL TO LOGICAL FORM	10
INFERENCING CAPABILITY	17
SMANTIC ISSUES	23
FURTHER SEMANTIC EXTENSIONS	24
SUMMARY	28
REFERENCES	29

LOGICAL INTERPRETATION OF RELATIONAL DATABASES

Ronald M. Lee

INTRODUCTION

In recent years, there has been a gradual convergence of interests between researchers in Database Management (DM) and Artificial Intelligence (AI). Broadly speaking, the principal concern of DM has been to *efficiently* maintain and retrieve large amounts of data in computer accessible storage. This data typically involves a large number of similarly structured records--e.g., purchase orders, sales transactions of a large company.

AI systems, by contrast, emphasize neither efficiency nor large data storage, but rather *versatility*— i.e., in dealing with complex, poorly structured problem areas such as pattern recognition, language understanding theorem proving, etc. These systems also typically involve a "data base," though of a different type than in DM: It is usually much smaller (almost always contained in main memory) and structurally more complex. Rather than dealing with character strings and numbers arranged in records and data files, a common form of an AI database is as statements in a logical predicate calculus. These provide the basis for the deductive capacity of these systems.

Our interest here is a combination of these two approaches: in the interpretation of DM databases as assertions in a logical syntax. This has two motivations: one, to provide an inferencing capability in queries to these databases, and two, to clarify the semantic issues in the way these databases are used to model organizational environments.

DATABASE INFERENCING

Using a new classic taxonomy of Anthony (1965), organizational activities may be divided based on the length of the time horizon their decisions affect: operational activities have a time framework in terms of days or weeks; tactical activities have an effect in terms of months to a year; strategic activities affect a number of years.

Using Simon's distinction of programmed vs. non-programmed activities (or structural vs. unstructural), based on whether the activity can be directed by an explicit procedure, Gorry and Scott Morton observe that as one moves from the operational to the strategic activities the associated decisions tend to become increasingly less structured.

The major use of DM databases to date has been in data processing applications. In the above framework, they are used mainly for structured, operational level activities such as sales order processing, billing, inventory control, etc. Therefore, the people that use them for the most part do so frequently and routinely.

These databases might also be useful in less structured, longer range activities, though the requirements in this case are somewhat different.

a. information is usually required in more summarized form
b. access is less routine–information must be retrievable in a variety of forms and combinations
c. the information often is used in combination with other informational and computational resources.

What we have just described is the use of a DM database in *decision support* applications. The principle point of the criteria we mentioned is that the data needs in these cases, though contained in the database, will often not be at the detail level nor in the structural arrangement in which the database was designed.

It is for these uses that a mechanism providing inferencing on the database is needed.

One obvious way of summarizing data is simple arithmetic calculations–e.g., counts of inventory, etc. What this lacks however is a corresponding framework of qualitative inferencing. For instance, if one has an inventory of three apples and two oranges and count them up, you have five "things," but what descriptive label should be attached to this broader class?

This is an example of where a system of qualitative inference would be useful. More realistic examples abound, e.g., in accounting data if you have $500 in cash and $700 in accounts receivable, then you have $1,200, but of what?

Conversely, one might have wished to make a query about the quick assets of the company when the database only contained data on cash and accounts receivable.

SEMANTIC ISSUES

Closely related to considerations of inferencing are those of the *semantics* of the database. A database, clearly, is a collection of symbols which are intended to 'stand for' objects and other environmental phenomena. What the individual symbols stand for, and more importantly, what combinations of these symbols stand for, is their semantics.

In routine data processing applications, the semantics of the data is generally not a problem. The people who use the system are thoroughly familiar with the vocabulary of symbols they use, and the interpretation of each within the limited decision context of the application. Thus, for instance, a clerk using an airline reservations system his little problem understanding the parameters involved–flights, seats, passengers, cities, etc.

On the other hand, when a database is used for decision support applications, one is much more likely to deal with aggregated concepts whose definition must be carefully constructed and communicated between system and user so as not to be mis-understood. For instance, a technical definition of PASSENGER PLANE may or may not exclude planes which do not have a separate cock-pit. In administrative contexts one encounters many such terms with exacting and often not obvious technical definitions, for instance, GROSS MARGIN, SEPARABLE MARGIN, CONTROLLABLE MARGIN, MARGIN NET OF OPERATING EXPENSES, MARGIN NET OF TAXES, MARGIN NET OF EXTRA-ORDINARY ITEMS, etc. etc.

Here a data dictionary facility would obviously be useful. However, from a decision support perspective this solves only part of the problem, namely describing the built-in calculations of the system.

However, what often happens is that none of these existing functions provide the specific calculation needed for a non-routine decision. For instance, a manager wants to evaluate a department's performance including the effect of an extraordinary expenditure net of tax.

This can of course be provided through re-programming, but it would be much more effective if the manager could express this request directly, explaining it in the conceptual terms in which he/she is accustomed.

For this the system needs not only to be able to do such calculations, but also to be able to reason about how such calculations are defined. For instance, in the above example it may not be possible to calculate the tax applicable to a single department since the tax for the entire company is in a different tax bracket than would be the tax for the department alone.

When attempting to describe the semantics of a database, we generally do it by means of another language, e.g., English. This is perfectly legitimate provided the user audience thoroughly understands and agrees upon the English explanation. When the concepts we are trying to explain are technical and complex, this may be unsatisfactory and we look for more structured and formal methods to explain the semantics.

Here again, the interpretation of a database in terms of a logical syntax will be useful since formal semantics has been studied chiefly in the context of logical languages.

OUTLINE OF PAPER

To provide a framework for discussion about DM databases, these are assumed to be structured according to the Relational Model (Codd 1970). While few commercial systems actually conform to this model, is serves as a convenient abstraction of their essential structural features. Furthermore, the relationship between the various popularly-used file and database management schemes (e.g., CODASYL) and the Relational Model has been well-studied (see e.g., Date (1977), for an overview).

As an interesting aside, these studies typically regard the Relational Model as the ultimate level of abstraction of computer-based data files. Here, by contrast, the Relational Model is used as the starting point for a considerably more abstracted view. Indeed, the view of "data" as character strings in a structured format will be re-interpreted as predicate names, etc. arranged by a logical syntax.

The Relational Model will be briefly reviewed. Next, we present (in summary form) a fairly standard form of the first order predicate calculus (FOPC), and extend it to a 'multi-sorted' form for use in describing databases. Following that, a translation from the relational syntax to the logical form is presented.

The potential for inferencing from this logical form is discussed and illustrated, and the semantic issues are examined.

REVIEW OF THE RELATIONAL MODEL

The Relational Model, in its essentials, is conceptually very simple. Basically, similarly structured facts about the organization are collected in columnar tables, called *relations*. The columns of these tables are given names, called the *attributes* of the relation. The rows of the table are called *tuples*, and each indicates a certain fact or information about a certain entity in the organization. A relation typically has one or more attributes designated as "key attributes," which uniquely identify the row, or correspondingly, the entity which the row describes.

For instance, consider the following relation describing vehicles.

VEHICLE	(ID	TYPE	COLOR	WEIGHT)
	A-27	TRUCK	RED	1200
	C-17	TRAILER	GREEN	300
	L-32	CAR	BLUE	700

Here, the relation name is VEHICLE, and it has four attributes, ID#, TYPE, COLOR and WEIGHT—i.e., these are the relevant characteristics about vehicles in this context. The relation has three types, indicating three vehicles, with the "values" of the attributes for each, listed in each row under the appropriate column. The attribute ID#, is a key—i.e., whereas more than one vehicle may be of the same type, color or weight ID# is unique for each vehicle.

In this example, each tuple represents an individual object. Relations may also be used to indicate associations between objects. For instance, suppose we had another relation, EMPLOYEE, which had row-wise description of the various employees as follows:

EMPLOYEE	(NAME	E-NUM	SKILL	SALARY)
	JONES	12-345	MANAGER	30000
	ADAMS	65-216	DRIVER	20000
	SMITH	28-159	CLERK	15000
	PERKINS	37-212	DRIVER	22000

Here the attribute E-NUM (employee number) is the key.

Then, the association between certain vehicles and certain employees, namely who drives which vehicle. This might be recorded as the relation DRIVERS.

DRIVERS	(V-ID	E-NUM)
	A-27	65-216
	A-27	37-212
	L-32	37-212

Here, V-ID refers to what was called ID in the VEHICLE relation. E-NUM here is the same as E-NUM in the EMPLOYEE relation. In the relation DRIVERS, both of these attributes together constitute the identifying key. At the same time, each separately is a "foreign key"—i.e., uniquely identifying rows in other relations.

The relation DRIVERS thus indicates that vehicle A-27 (the truck) is driven by employee 65-216 (Adams) and employee 37-212 (Perkins). Vehicle L-32 (the car) is driven only by employee 37-212 (Perkins).

It should also be mentioned that another important aspect of the Relational Model derives from the simplicity and power of the various access languages that have been defined for it, especially the relational calculus and the relational algebra. (See e.g., Codd (1970)). As our purposes are rather different, these languages will not be discussed here.

PREDICATE CALCULUS--REVIEW AND NOTATION

Notation as Applied to Physical Objects

It is assumed that the reader is at least generally familiar with the predicate calculus and its syntax. The following is thus only a review.

The description of a logical system begins by declaring its *universe of discourse.* This is an informal (meta-language) description of the types of objects that the statements in the logic are about.

For the moment, we will assume that the individuals described by the logic are identifiable physical objects at a point in time. We will return to examine this aspect more closely later in the discussion of semantic aspects.

In summary form, the basic constructs of a first order predicate calculus are as follows:

1. *Propositions.*
   These are complete logical statements having a truth value. These are indicated symbolically by capital letters--e.g., P,Q,R.
2. *Logical connectives.*
   These combine propositions to form new logical statements, also having a truth value.
   The logical connectives used here are as follows:

$\leftrightarrow$	equivalence
$\rightarrow$	implication
&	conjunction
V	disjunction (inclusive)
W	disjunction (exclusive)
$\sim$	negation

3. *Individual constants and variables.*
These stand for objects in the domain of discourse—e.g., individual trucks or employees.

Individual constants are denoted as one more lower case letters, possibly containing non-leading digits or hyphens; e.g., a, bill, truck-7.

Individual variables are denoted by a "?" followed by one or more capital letters or digits, e.g., ?X, ?Y2.

4. *Functions.*
These map one or more individuals to another—e.g., supervisor (jones) refers to another individual who is Jones' supervisor. Functions may take zero or more arguments and always result in a reference to a single individual. Functions may thus appear wherever an individual constant is allowed. Indeed, a zero-place function is the same as an individual constant. Functions for physical objects are therefore denoted in the same way as individual physical constants, but followed by an argument list, e.g, f(a), boss(smith)

5. *Predicates.*
These indicate features, properties, attributes, etc. applied to zero or more individuals. Predicates will be denoted by upper case letters or words, e.g., P(?X), RED(?X), OWN(x,y). When a predicate is applied to individual constants or to quantified (see below) individual variables, or to functions of these, it has a truth value and may be combined to form other logical statements using the logical connectives above. A zero-place predicate is equivalent to a proposition.

6. *Logical quantifiers.*
These indicate the range of individual variables. The principal ones are:

$\forall x$ universal quantifier
(for all x, for each x,—
ranging over all individuals
in the universe)

$\exists x$ existential quantifier
(for some x—ranging over
at least one individual)

Parentheses are used in the usual fashion.

A MULTI-SORTED LOGIC

In order to effectively capture the modeling aspects of a relational database, we will essentially combine three logics of the above form. This is called a 'multi-sorted' logic, which basically uses the same logical mechanisms as the FOPC, with additional syntactic constraints.

The universe of the first logic will be entities, assumed as above to consist of identifiable physical objects.

The next universe consists of what computer people refer to as "character strings"--i.e., unambiguous sequences of alpha numeric symbols and punctuation. Normally in this work the name of a particular property or association is not important--it is the relationship of this property to others (indicated by logical theorems and axioms) which is most relevant. However, in certain limited cases, the actual spelling does matter. This occurs in what we call labels attached to an individual, such as a persons name or social security number, the license number of a car, the serial number of a piece of equipment or the street address of a building. These are all uses of character strings where the characters themselves do not convey any meaning but are used purely to distinguish this object from others similar to it.

The other category of objects to be recognized are the *real numbers* (including obviously the integers as a special case). These are needed to reflect properties recorded as numeric measurements, a very important aspect in administrative contexts.

The proposed universe of discourse no doubt seems to be a rather odd mix, consisting of character strings, numbers and physical objects. However, while these three types of objects interact, they may be viewed in terms of *three separate universes* having distinct logical operations defined on them. In addition to providing a convenient logical organization, this also provides a useful semantic division: the first two universes, character strings and numbers, consist of "representational objects," whereas the last consists of "real world objects." This division is perhaps the most fundamental point of this paper: distinguishing those aspects of a database which are internal representational from those that pertain to the external environment.

The additional notation for these latter two universes is as follows:

Special Notation for Character Strings

1. *Character string constants.*
   These are any sequence of zero or more letters, digits or punctuation enclosed in double quotes. (Two double quotes are used to represent the double quote itself.) e.g., "apple", "#27-512", "T.S. Eliot".
2. *Character string variables.*
   These are one or more capital letters followed by a "$," e.g., A$, XY$.

3. *Functions.*
Note—these map from any type of individual to a character string.
Notation--one or more capital letters followed by a "$" and an argument list.
E.g., FIRST-LETTER$("APPLE"), LAST-NAME$(bill).

A special fix function, "+", is used for concatenation--e.g., "AP"+"PLE".

4. *Predicates.*
The only character string predicate used is "=". This is presumed case insensitive, e.g., "LEE" = "Lee" = "lee".

This is typically used to relate a character string function and a constant, e.g., LAST-NAME$(bill) = "SMITH".

Special Notation for Numbers

1. *Numeric Constants.*
   a. real numbers--a sequence of digits with embedded decimal point and optional sign, e.g., 1.2 -3.75 6.0
   b. integers--a sequence of digits with no decimal point, and with optional sign, e.g., 1, -3
2. *Numeric Variables*
   a. real numbers--one or more capital letters suffixed by "#" e.g., A#, XY#
   b. integers--one or more capital letters suffixed by a "!" e.g., A!, XY!
3. *Numeric Functions.*
In general, these may map from any type of individual(s) to a number.
Notation:
   a. Real numeric functions--one or more capital letters suffixed by a "#" and followed by an argument list, e.g., WEIGHT#(bill)
   b. Integer functions--one or more capital letters suffixed by a "!" and followed by an argument list, e.g., POPULATION!(austria).

Additionally, the usually infix arithmetic functions are assumed which map from reals to reals or integers to integers: +, -, *, /, **.

4. *Numeric predicates*
The following commonplace numeric predicates are assumed:

= equals

≠ not equals

$>$ greater than

$\geq$ greater than or equals

$<$ less than

$\leq$ less than or equals

These will often be used to relate a numeric function to a numeric constant or arithmetic expression: e.g., WEIGHT#(bill) < WEIGHT#(sally) + WEIGHT#(mary).

NOTE ON CONTROL IMPLICATIONS

In the pages to follow, the processing implications of the aforementioned separation of universes should be kept in mind. This enables a separation of labor between the components dealing with purely representational facts versus those involving deductions about the external environment. Thus, as assertions or queries are made, one can imagine a first pass made by a character string processor (a la SNOBOL) which resolves concatenations and predicates involving only character strings. A second pass would be made by a numeric processor, which did the purely numeric calculations. Lastly, deductions on the database itself are made.

TRANSLATION FROM RELATIONAL MODEL TO LOGICAL FORM

The translation of relational databases into the predicate calculus will be illustrated using the example databases used earlier. For convenience, these are repeated in Exhibit 1 along with their logical reformulation, explained in the pages to follow.

First, note that for certain relations e.g., VEHICLE, EMPLOYEE, each row (tuple) of the relation implicitly signifies the existence of a different individual in the universe. We will make this aspect explicit by the introduction of logical constants corresponding to each one of these rows. For notational simplicity we here make use of simple alphabetic letters "internal" names for these individuals. In a computer implementation, these would be "system-generated identifiers" (e.g., GENSYM of LISP), providing a unique internal identification for each of the physical objects acknowledged in the system.

Note further, that not every relation indicates the existence of individuals corresponding to its tuples--e.g., the rows of the relation DRIVES do not correspond to the existence of additional individuals, merely associations between already recognized individuals.

Thus in our simple database, internal names are need for the three vehicles, say a, b and c, and for the four employees, say d, e, f and g.

Next, we note that the relation name itself indicates a property of these individuals--namely that the first group are vehicles, the second employees. Thus we have:

VEHICLE(a)

VEHICLE(b)

VEHICLE(c)

EMPLOYEE(d)

EMPLOYEE(e)

EMPLOYEE(f)

EMPLOYEE(g)

As was alluded earlier, the key attributes for these relations will be interpreted here as labels. Adding this to the previous statements, we have

VEHICLE(a) & ID$(a) = "A-27"

VEHICLE(b) & ID$(b) = "C-17"

VEHICLE(c) & ID$(c) = "L-32"

EMPLOYEE(d) & E-NUM(d) = "12-345"

EMPLOYEE(e) & E-NUM(e) = "65-216"

EMPLOYEE(f) & E-NUM(f) = "28-159"

EMPLOYEE(g) & E-NUM(g) = "37-212"

Looking specifically at the relation VEHICLE, let us consider next the attribute designated as TYPE. We see that the values of this attribute are other predicate names, on a par with the relation name. E.g., we have for the object a:

VEHICLE(a) ID $(a) = "A-27" & TRUCK(a).

That is, both VEHICLE(a) and TRUCK(a) are single place predicates indicating features about the object a. It is interesting that in this logical interpretation of relational databases, ***relation names and attribute values have the same status***, i.e., as predicate names. (This statement must be qualified for the cases where the attribute values are labels or

numbers. However, as stated above, these can be regarded as an abbreviated form for further predicate names.)

In this case, the name of the attribute itself, namely TYPE, seems to offer little information--it tells us little about the sorts of terms (predicates) that can appear in this column--e.g., could not BIG, SMALL, AQUATIC, AIRBORNE, etc. also be regarded as TYPEs of vehicles?

On the other hand, the next attribute COLOR, does indicate something about the range of terminology that can appear--e.g., we know that YELLOW, BLACK, etc. are possible colors, where as BIG, SMALL are not. Now, by the preceding method, we consider the attribute values here as predicates--e.g.,

RED(a)

GREEN(s)

BLUE(c)

How then do we interpret the word COLOR? Our approach is to regard this as what might be called a ***predicate group***. COLOR(?X) is in effect a second order predicate, applying not to individual objects, but to predicates--e.g., COLOR(RED) is true, COLOR(BIG) is false. However, recognizing second order predicates complicates the logic considerably, something we want to avoid if possible. Note that the principal purpose of this predicate is to enumerate the list of possible color predicates. In addition, since in the relational model attributes are single-valued, choosing COLOR as an attribute name implicitly indicates that these various color predicates will be mutually exclusive. These features can be incorporated into a first-order predicate, call it HAS-COLOR (?X) with an axiom of like the following.

$$\text{HAS-COLOR}(?X) \leftrightarrow \text{WHITE}(?X) \;\; W$$
$$\text{YELLOW}(?X) \;\; W$$
$$\text{RED}(?X) \;\; W$$
$$\text{etc.}$$

That is, any object satisfying HAS-COLOR(?X) will satisfy one and only one of the listed color predicates.

Thus, in the logical interpretation, attributes of this type and their values are seen as corresponding to two one-place predicates, the first being the name of the attributed renamed with the prefix "HAS-," the other being the attribute value. However, since these two predicates are so closely related, and for notational simplicity and readability, we adopt a notational convention resembling that of the functional form--with the (un-prefixed) attribute name in the position of the function name, and the

attribute value in the position of the function value, e.g.,

$$\text{COLOR}(?X) = \text{RED}$$

It is to be emphasized that this is a notational convenience only. In expanded form this is equivalent to:

$$\text{HAS-COLOR}(?X) \;\&\; \text{RED}(?X).$$

The last attribute of the VEHICLE relation is WEIGHT. The values of this attribute are numbers indicating the vehicle weight, presumably in kg.

In the logical reformulation, attributes whose values are numeric are treated as *measurements*. This involves a mapping from the object to the real or integer numbers. In the theory of measurement, such mappings involve two components: a *measurement operation* which is the process of measuring the objects properties, such as weighing the object or measuring its length, and a *measurement standard*, such as a standard kilogram or standard meter. As is well known, such measurement standards are established as the properties of certain unique physical objects maintained by such institutions as the U.S. National Bureau of Standards. Practically, these standards are communicated by replicas of these standard objects--e.g., sets of brass weights, rulers, etc. In ordinary commercial practice, these are considered equivalent for measurement purposes--that is, one typically is not concerned which ruler was used to measure an objects length. Indeed, except in such exacting areas as physics, etc. the process of measurement has become so commonplace that the nature of the measurement operation and how it is ultimately related to the unique object representing the measurement standard are often ignored. We thus use such terms as kilogram, pound, meter, foot, etc. as if they were objects in themselves. For instance, in such statements as "X weights 10 kilograms," we accept the notion of a kilogram as something familiar to everyone reading this statement, and not dependent on particular measurement devices, the measurement procedure used and so on.

To express measurement in this logical framework, however, we strike something of a compromise between the theoretical distinctions of measurement operations and measurement standards and the ordinary views of measurement. In doing this, we introduce in our universe additional objects called "measurement standards," giving them lower case names for individual constants such as "kilogram," "pound," "meter," etc. Technically, we may assume these to be the standard objects such as are maintained by the Bureau of Standards, or, more in line with common usage, we may regard them as abstract objects.

The measurement operation is subsumed in the definition of certain *measurement predicates* or *measurement functions*.

A measurement is here viewed as a three place relationship between a physical object, a measurement standard, and a number. For instance, the sample database, that the truck (constant name *a)* weighs 1200 kg is shown as the following predicate:

WEIGHT(a, kg, 1200)

However, since the number involved in such measures is usually unique (i.e., an object has only one weight in kilograms), this is usually shown in functional form, e.g.,

WEIGHT#(a, kg) = 1200.0.

At this point the basic constructs of converting from the representation of physical objects and their features in the Relational Model to a logical notation. This was illustrated using the relation VEHICLE in the earlier example database. The conversion of the next relation, EMPLOYEE, would be done in an analogues manner: the relation name EMPLOYEE becomes a single place predicate. The attributes NAME and E-NUM are label functions. The attributes SKILL is a predicate group consisting of such one place predicates as MANAGER(?X), DRIVER(?X), CLERK(?X), etc. Finally, the attribute HEIGHT is re-interpreted as a measure function.

We so far have not explained the last relation, DRIVES. This, it turns out is re-interpreted somewhat differently than the preceding two relations. Here, the relation name itself becomes a *two*- place predicate, applying to individuals vehicles and employees. Furthermore, the attributes in this relation, V-ID# and E-NUM, served only as foreign keys, e.g., to reference other tuples, and add no descriptive information of their own. They are therefore ignored in the logical interpretation. The logical formulation of DRIVES thus becomes:

DRIVES(a,e)

DRIVES(a,g)

DRIVES(c,g).

The complete database, in its original relational form and its re-interpretation in logical form is shown in Exhibit 1.

Exhibit 1: EXAMPLE DATABASES

RELATIONAL FORM:

VEHICLE	(ID,	TYPE,	COLOR,	WEIGHT)
a	A-27	TRUCK	RED	1200
b	C-17	TRAILER	GREEN	300
c	L-32	CAR	BLUE	700

EMPLOYEE	(NAME,	E-NUM,	SKILL	HEIGHT
d	JONES	12-345	MANAGER	1.91
e	ADAMS	65-216	DRIVER	1.72
f	SMITH	28-159	CLERK	1.83
g	PERKINS	37-212	DRIVER	1.78

DRIVES	(V-ID,	E-NUM)
	A-27	65-216
	A-27	37-212
	L-32	37-212

Exhibit 1 continued

LOGICAL FORM

VEHICLE(a) & ID$(a) = "A-27" & TRUCK(a)& COLOR(a) = RED & WEIGHT#(a,kg) = 1200.

VEHICLE(b) & ID$(b) = "C-17" & TRAILER(b) & COLOR(b) = GREEN & WEIGHT#(b.kg) = 300.

VEHICLE(c) & ID$(c) = "L-32" & CAR(c) & COLOR(c) = BLUE & WEIGHT#(c,kg) = 700.

EMPLOYEE(d) & NAME$(d) = "JONES" $ E-NUM$(d) = "12-345" & SKILL(d) = MANAGER & HEIGHT#(d) = 1.91.

EMPLOYEE(e) & NAME(d) = "ADAMS" & E-NUM$(e) = "65-216" & SKILL(e) = DRIVER & HEIGHT#(e) = 1.72.

EMPLOYEE(f) & NAME$(f) = "SMITH" & E-NUM$(f) = "28-159" & SKILL(f) = CLERK & HEIGHT#(f) = 1.83.

EMPLOYEE(g) & NAME$(g) = "PERKINS" & E-NUM$(g) = "37-212" & SKILL(g) = DRIVER & HEIGHT# = 1.78.

DRIVES(a,e)

DRIVES(a,g)

DRIVES(c,g)

INFERENCING CAPABILITY

Simply converting databases from a relational form to a logical one is not in itself very valuable. It must be shown how databases re-cast in this form can be more useful.

Note, first, that the fact that the logical form appears more verbose is not a serious objection. Our concern here is re-formulation at a *conceptual level*. At the machine implementation, *storage level*, more compact notations can be used, indeed perhaps making use of the original storage form with certain auxiliary markers added which indicate its conversion to logical form.

Second, and more importantly, observe that the conversion from relational to logical form is *not* a mechanical one. In doing the conversion, we made use of background knowledge about the environment that the database described. This is a fundamental point: the logical form encodes more semantic information. For instance, in doing the conversion we specified the relations whose tuples indicated the existence of individual objects in the environment. Further, the interpretation of attributes distinguished between labels, predicate groups and measurements, additional semantic information.

The inferencing capability available from a database re-interpreted in this way is perhaps best presented in terms of the three universes of discourse involved: character strings, numbers and physical objects.

On the other hand, inferences within the first two domains are well-studied, and have been implemented as primitive operations in a number of programming languages (e.g., SNOBOL, text editors for character strings; FORTRAN, PL/I, etc. for numbers). Our attention will therefore be to inferences within the universe of physical objects, and the relationship of this domain to the other two, i.e., labels and measurements.

The general rules of inference for first order predicate logic has also been well studied.

Inferences Involving Only Physical Object Individuals

In the example databases earlier, their logical translation consisted entirely of conjuncts of predicates (including labels, measurements) applied to individual constants. This is a general feature of the translation of relational databases--they consist mainly of specific, conjunctive facts. For instance, it is very seldom that disjunctive or negative facts appear--e.g., "the truck is either blue *or* brown" or "the truck is *not* green." This types of statements can be viewed as partial facts that is, one does not know the color of the truck exactly, but nonetheless, something is known about the color. Such partial information appears very seldom in administrative databases. They consist, by contrast, of complete facts.

At the same time, such databases can be viewed as consisting of elementary facts from which more complex statements can be constructed. This is indeed one of the earliest design principles in data processing: for flexibility in information reporting, one should collect and store the data in its most "disaggregated" form. Thus "aggregation," or deduction based

on the elementary facts in the database, has always been an objective of file and database systems. The *rules* for these deductions were however implicit in the minds of the programmers and end-users who specified the retrieval. The retrieval programs or queries had to be stated in terms of the elementary relation and attribute names and their values.

Given the logical re-interpretation of these databases, one can make certain of the more frequent or more complex deductions explicit as axioms about the database.

Retrieval requests (including "alterters," stored queries which trigger on an exception basis) can be stated in terms of higher level, derived facts.

One of the simplest and perhaps most useful types of inferences is for hierarchies of classification, what have come to be called "generalization hierarchies." These were first proposed in the database management literature by Smith and Smith (1977), though they were discussed in Artificial Intelligence some years earlier, e.g., Quillian (1968). A generalization hierarchy is a graphical representation of a sequence of subset relationships between categories. To illustrate, the Smiths' (1977:109) coincidentally also made use of the example of vehicles. Two of their graphs are reproduced in Exhibit 2.

The arcs in such generalization hierarchies are often read "is a." Thus an air vehicle "is a" vehicle, a plane "is a"(n) air vehicle, a passenger aircraft "is a" plane, etc.

These generalization hierarchies translate into logical assertions in a straightforward fashion, namely as simple implications, e.g.,

$$\text{All(?X):AIR-VEHICLE(?X)} \rightarrow \text{VEHICLE(?X)}$$

$$\forall\text{(?X):PLANE(?X)} \rightarrow \text{AIR-VEHICLE(?X)}$$

$$\forall\text{(?X):PASSENGER-AIRCRAFT(?X)} \rightarrow \text{PLANE(?X)}$$

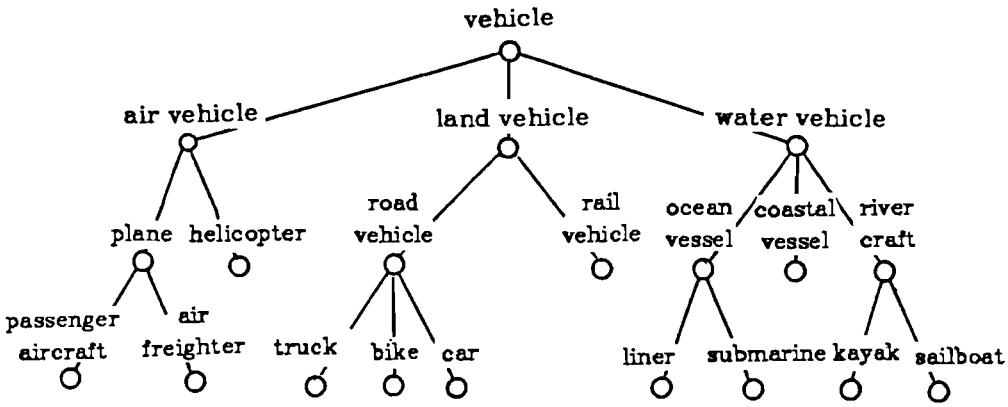
However, in the Smiths' representation, additional information is meant to be conveyed, namely that each of subsets linked to a common parent node is mutually exclusive.

If the subsets are also exhaustive, this translates into the logical syntax as exclusive disjunction and equivalence, e.g.,

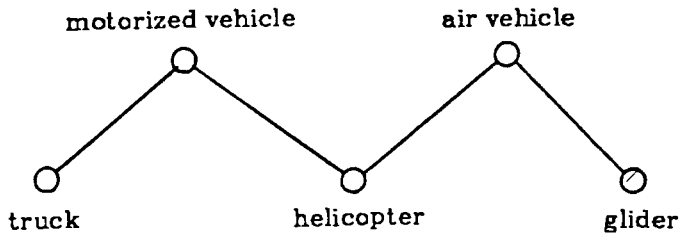
$$\begin{aligned} \text{VEHICLE(?X)} \leftrightarrow\ & \text{AIR-VEHICLE(?X)} \quad \veebar \\ & \text{LAND-VEHICLE(?X)} \quad \veebar \\ & \text{WATER-VEHICLE(?X)} \end{aligned}$$

If the subsets are mutually exclusive but not exhaustive (a distinction which the Smiths' graphs don't distinguish), we need a way of referring to the possible residual category. As a convenience, a variable predicate, ?OTHER( ) is used for this purpose, e.g.,

Exhibit 2. GENERALIZATION HIERARCHIES



a. A generic hierarchy over vehicles



b. A generic hierarchy which is not a tree

$$\text{VEHICLE}(?X) \leftrightarrow \text{AIR-VEHICLE}(?X) \ \text{W} \ \text{LAND-VEHICLE}(?X) \ \text{W} \ \text{WATER-VEHICLE}(?X) \ \text{W} \ ?\text{OTHER}(?X)$$

Graph b. of Exhibit 2 illustrates the possibility of set intersection, i.e., that a helicopter is both a motorized vehicle and a land vehicle. In a logical syntax, this is expressed as a conjunction, i.e.,

$$\forall(?X): \ \text{HELICOPTER}(?X) \rightarrow \text{MOTORIZED-VEHICLE}(?X) \ \& \ \text{AIR-VEHICLE}(?X).$$

These graphical representations have a clear superiority over the logical form in terms of pictorial clarity (though this diminishes rapidly for graphs with a large number of nodes and arcs).

The over-riding advantage of the logical form is, however, its robustness. The generalization hierarchies are but suited for showing the interdependence of properties of single objects, but break down when multiple object inter-dependencies are involved. suppose for instance, we considered a relationship LICENSED(?X,?Y), which indicates that a person, ?X, is licensed to drive vehicle ?Y. As is the case in many states in the U.S., a special chauffeur's license is required to drive a bus, in addition to the ordinary driver's license. Thus, a person who may drive a bus may also drive a car, but the reverse is not true. This is expressed logically as follows:

$$\forall(?X) \exists(?Y) \ \text{LICENSED}(?X,?Y) \ \&$$

$$\text{BUS}(?Y) \rightarrow \forall(?Z) \ \text{CAR}(?Z) \rightarrow \text{LICENSED}(?X,?Z).$$

Reading: For any person, X, if x is licensed to drive some bus, then x is licensed to drive any car.

Additionally, derivative properties can be defined based on participation in two place relationships. For instance, a bus driver is one who is licensed to drive a bus:

$$\forall(?X) \ \psi e \ (?Y): \ \text{LICENSED}(?X,?Y) \ \& \ \text{BUS}(?Y) \rightarrow \text{BUS-DRIVER}(?X)$$

The general organization of this inferencing structure should now begin to become evident. Basically, four types of knowledge are involved. The first two consist of the general purpose axioms and inference rules of first order predicate logic. The second two are developed for the particular environment being modeled and consist of "facts" from the database, and generalizations about how to convert these facts into others. The facts, as was seen, are statements which are generally conjuncts of predicates applied to individual constants. The generalizations, on the other hand, are typically universally quantified assertions involving variables. Inferencing thus amounts to the application of these universal statements to the particular fashion the database.

Inferences Involving Labels
(Physical Objects and Character Strings)

Remember that a label was defined as a character string associated with an object for identification purposes only. It carries no additional information about the properties of the object. The amount of inferencing involving labels is therefore fairly limited.

While labels are used for identification, there is no a priori assumption that a particular label identifies a unique individual, for instance, many people may have the same last name. For those types of labels that are uniquely designating within a given type of object, this may be stated as an explicit axiom. For instance, in the earlier example database, the label function ID$ had this property. This is stated:

$\forall$(?X) $\forall$(?Y): [VEHICLE(?X) & VEHICLE(?Y) & ID$(?X) & ID$(?Y) $\rightarrow$?X = ?Y.

Reading: for any two vehicles, x and y, if they have the same ID$, they are the same object.

Unique identification is not always determined by a single label, but sometimes by a conjunction of more than one. For example, an office might be uniquely identified by a building code, BLDG$, and an office number, O-NUM$:

$$\forall(?X)\ \forall(?Y)\text{: [OFFICE(?X) \& OFFICE(?Y) \&}$$

$$\text{BLDG(?X) = BLDG(?Y) \&}$$

$$\text{O-NUM\$(?X) = \$ O-NUM\$(y)]} \rightarrow \text{?X = ?Y}$$

The values of label functions may interact in other ways with other predicates. For instance, to state that any two people that are married have the same last name:

$\forall$(?X) $\forall$(?Y): MARRIED(?X,?Y) $\rightarrow$ LAST-NAME$(?X) = LAST-NAME$(?Y).

Inferences Involving Measurements (Physical Objects and Numbers)

Numeric measurement is an extremely important type of data in administrative environments and there are numerous types of inferences that are useful. Indeed many numeric algorithms of management science are essentially inferences based on numeric measurement of physical objects.

A basic type of inference is conversion of units of measure. Recall that units of measure appeared in the syntax here as special individual constants marked with the prefix "@." Thus, conversion from weight in pounds (@lb) to kilograms (@kg) is as follows:

$$\forall(?X): \text{WEIGHT\#}(?X,@kg) = 1/2.19 * \text{WEIGHT\#}(?X,@lb)$$

Similarly, conversion from length in feet (@ft) to length in meters (@m) is stated

$$\forall(?X): \text{LENGTH\#}(?X,@m) = 3.28 * \text{LENGTH\#}(?X,@ft).$$

conversion from such elementary measures to derived ones can also be stated. For instance, to calculate the volume of a cube in cubic centimeters (@cm3),

$$\forall(?X): \text{CUBE}(?X) \rightarrow$$

$$\text{VOLUME\#}(x,@cm3) = \text{LENGTH}(?X) * \text{WIDTH\#}(?X) * \text{HEIGHT\#}(?X).$$

A very important type of measurement in administrative contexts are monetary valuations. For instance, the historical cost of the object, a, in dollars might be

$$\text{HIST-COST\#}(a,@\$) = 1500.00.$$

More recent views in accounting, so-called "price-level accounting," recognize changes by year in the standard dollar used for valuation. Thus the value of an object in, say, 1962 dollars is not the same as its value in 1978 dollars. To indicate this conversion, e.g., recognizing 40% inflation, we could state:

$$\forall(?X): \text{HIST-COST\#}(?X,@\$\text{-}1972) = 1.40 * \text{HIST-COST\#}(?X,@\$\text{-}1962).$$

Accounting provides a wealth of opportunities for inferences based on numeric measurement. Indeed, an eventual objective of this line of research is to represent accounting definitions and the varieties of manipulations (e.g., FIFO vs LIFO inventory valuations, depreciation methods, overhead allocations, variance analysis, etc.) in terms of the logical calculus. This would enable the embedding of accounting knowledge in the system itself, so that it could aid in revising calculations based on different assumptions and decision criteria. However, the more interesting

and challenging aspects accounting problems inevitably involve inferences across time. Thus a fuller treatment of accounting is differed to a later paper.

SEMANTIC ISSUES

As indicated in the introduction there were two goals of interpreting a DM database in logical form: inferencing, which was just discussed, and clarification of the semantics of the database, which we consider now.

A (DM) database is, as observed earlier, a type of formal language in itself—i.e., it is composed of symbols which are manipulated according to certain rules. Also as mentioned above, a formal language has two components: its *syntax*, giving the vocabulary and rules describing permissible combinatons of this vocabulary, and its *semantics*, indicating the phenomena which these expressions represent.

A principle shortcoming of the formalisms for DM databases, such as the network and relational data models, is that they make explicit the syntax of the database, but not its semantics.*

Simply translating the syntax of a database into a logical syntax does not however resolve these semantic issues, but it does help to focus them.

The concept of semantics we are using here is called *model theoretic* semantics or denotational semantics, and is due originally to Tarski (1931 and 1956).

MODEL THEORETIC SEMANTICS

The basic concept is that the references of the symbols in the formal language are explained in terms mathematical sets of objects, called a *model* of the formal language. The definition of these sets is agreed upon extra-logically, that is outside of the formal language.

One important set is called the set of truth values, V:

$$V = \{True, False\}.$$

The denotation of a proposition or sentence is an element of V.

In discussing the FOPC language, we introduced another set, the *universe of discourse*. We will call that set E, and as before assume that this is the set of physical entities at a given time.

The individual constants of the FOPC denote elements of E.

The one place predicates of the FOPC denote subsets of E.

---

* Clarification: "semantics" is sometimes used with regard to programming and query languages to indicate *what the computer does* in response to a symbolic command. Here we mean "semantics" in the linguistic sense—what the symbols in the database refer to in the external environment.

The 2 to n place predicates of the FOPC denote two through n place relations on E.

The 1 to n place (logical) functions of the FOPC denote one to n place (mathematical) functions on E.

The variables of the FOPC range over the elements of E.

The quantifier $\forall u\ \Phi(u)$ (for some variable u and formula Φ) is interpreted as a 'big conjunct' of propositions formed by substituting a constant representing each element in E for u, i.e.,

$$\forall u\ \Phi(u) ::= \Phi(a)\ \&\ \Phi(b)\ \&\ \Phi(c)\ \&...$$

where a, b, c, ... are names of elements of E.

The qualifier $\exists\ u\ \Phi(u)$ is correspondingly interpreted as a 'big disjunct,' i.e.,

$$\exists u\ \Phi(u) ::= \Phi(a)\ V\ \Phi(b)\ V\ \Phi(c)\ V\ ...$$

The semantics for the FOPC applying to physical objects can therefore be described as the model (M):

$$M = <V,E,F>$$

where V is the set of truth values and E is the set of (physical) entities. The symbol F is called an interpretation function. Its domain is the set of symbols in the formal language and its range is the sets V,E. F therefore provides the mappings from the individual constants, predicates, etc. to elements, subsets, and relations on V and E.

Following the presentation of the FOPC, we extend this to a 'multi-sorted' logic, adding the sets of character strings and numbers to the universe of discourse. Calling these sets respectively C and N, the semantic model now takes the form:

$$M = <V,E,C,N,F>.$$

Now individual constants refer to any element of E, C, or N, and predicates refer to relations on the union of E, C, N.

FURTHER SEMANTIC EXTENSIONS

This paper has dealt with the introductory aspects to a broader formal language, called CANDID (Lee 1980, 1981b), aimed at formal description of phenomena in administrative environments, again with the objective of providing a formal inferencing framework for managerial decision support applications.

We here consider in summary form some additional semantic considerations treated in more depth in that work.

1. Non-individualized Objects

The predicate calculus framework adopted here presumes that each object in the universe can be individually named by a constant. This causes no problems when dealing with 'middle-sized' objects such as persons, vehicles, machines, etc. However, organizational environments are also filled with smaller objects such as pencils, screws, nuts and bolts which, while counted, are seldom named individually.

At an even lower level, granular solids, such as corn, wheat are also commonplace, as are liquids, e.g., oil, water and even gases, e.g., propane.

From a theoretical perspective, the formal treatment of such non-individualized objects can be quite problematic. However, if we consider the practical requirements in organizational settings, the problem is quite simplified. The point is of course that organizations do not treat these objects at this small scale, but invariably raise them to a middle-size scale using the device of a *container* –e.g., a carton, drum, boxcar, tanker, etc. which are in turn individually identified.

The contents of such containers is specified not by an enumeration of individuals, but by a generic predicate describing the type of individuals contained (e.g., bolts, corn, propane) and a measurement predicate indicating the size of the population it contains, e.g., as a simple count or by other measures such as weight or volume.

The key point is that such individuals are regarded as inter-substitutable in these environments and their separate identities do not matter.

2. Time

Typically a DM database provides a sort of 'snapshot' model of the organizational environment–i.e., the facts in the database are presumed to be currently true in the environment.

An obvious extension to the semantic framework is to introduce the dimension of time, denoted say by the set T. Thus the

$$M = \langle V,E,C,N,T,F \rangle$$

But we must be explicit as to which times are included in the set T. Are this to be discrete units of time, e.g., days, or is time to be continuous, consisting of infinitessimally small time points. The latter seems to be the more descriptive, but again we confront the problem of how these are individually named. However a practical solution is available, if we again consider the requirements in organizational environments. The situation, as it turns out, is somewhat analogous to that with granular and liquid objects.

In organizations, it is difficult to choose any given time interval as an adequate minimum. For most sales transactions, accuracy to a day is probably adequate, but, for instance on the international stock and money markets, transactions must be timed to the split second.

Thus, while the conception of time in organizations is essentially continuous, the identification of individual times is invariably made referring to larger time chunks, what we call *time spans*. The Gregorian calendar provides individual names for time spans down to the level of a day—e.g., 12 October 1979, 18 May, 1990. Smaller time span units are provided by a 24 hours division of each such day (relative to a given time zone), and further divisions are available as minutes, seconds, milli-seconds, etc. Certain time points can be identified as the beginning and end points of such time intervals. For instance,

14:56:32 12 July 1980, Greenwich Mean Time

names a particular time interval of one second. Two time points are the beginning and ending of that internal.

Such a scheme does not name all the time points on a continuous time dimension, but on the other hand identifies all those that are ever needed for describing commercial and financial activities.

A second problem regarding time is the length of the time dimension. If we consider the typical usage of DM databases, it is generally accepted that they contain 'facts'—i.e., descriptions which are true of the environment.

Normally, in organizational contexts, assertions which pertain to the future are not considered factual, but rather are *conjectural* (or probabilistic) as with predictions; *intentional*, as with plans; or *obligational*, as with contractual promises.

This suggests that the time dimension T in the semantic model of a database contains only the times of the past up to the present. (Here we have another problem for the present is a moving point. Let us therefore consider only the semantic model of a snapshot of a database, for which the time point "present" is fixed.)

By adding the time dimension, the predicates reflected in the database, which formerly were interpreted as relations on E X C X N are now relations on E X C X N X T. That is, they may have additional places referring to times reflecting that they may be true of certain times, but not of others.

This however raises a new issue with regard to the other sets E, C, and N. The definition of the sets C and N was in terms of symbolic objects, hence independent of time. However, the set E was defined as the set of physical entities existing at some specified time.

One particularly important temporal aspect of organizational activity, especially in manufacturing, is that certain objects (products) are created while others (materials) are destroyed. This suggests that we must explicitly recognize existence as a predicate in the formal language. Correspondingly, the definition of the set E must be modified to include not just presently existing individuals, but any previously existing individuals as well.

3. Non-Physical Objects

So far we have maintained the assumption that the set E, of entities, comprises only identifiable physical objects. The reason for this, as discussed earlier, is that physical objects provide a reasonably uncontroversial basis for consensual identification of individuals. That is to say, if the formal language we construct is to achieve clarity and exactness of communication between user and machine, and via the machine between a larger community of users, it is essential that the elementary objects described in the language can be mutually and unambiguously identified by all parties involved.

However, this may seem to exclude a wide number of fundamentally important, though not physical, objects in administrative life. For instance, budgets, bank accounts, insurance policies, etc. are all examples of non-physical objects.

These might be viewed as special cases of a still broader class of abstract objects including for instance colors, theories, emotions, etc. The problem with admitting such objects into the formal language is that one loses all consensus of identification of individuals. (This has been a philosophical chestnut since the time of Aristotle. Strawson (1963), provides a useful discussion.) For example, contrast the questions: "How many children did Einstein have?" and "How many theories did Einstein have?" Presumably, an arbitrary group of people could agree on the answer to the first question given sufficient evidence. For the second, however, one would expect more disagreement, even given great amounts of factual material.

However, given the administrative orientation of the formal languages under discussion here, we can give a formal treatment to certain apparently abstract objects without 'throwing out the baby with the bathwater.'

Among these objects would be such examples as

accounts receivable
accounts payable
employment contacts
sales contracts
loans
bonds
stocks
insurance policies
budgets
etc.

A common feature in these kinds of objects is that they each contain an underlying aspect of obligation or permission. These are concepts formalized in a so-called *deontic logic* (see, for instance, von Wright 1968.) The extension of deontic logic to the modeling of administrative phenomena is discussed in detail in Lee (1981a).

A basic feature in the description of such commitments is that they are a relationship between two parties that involves a scenario of some future state of the world (i.e., describing the promised actions on both sides). Further, there may be alternative scenarios, as is the case with

contingent commitments (e.g., insurance) and penalty clauses.

The formal construct for dealing with this of description is called in philosophy and linguistics a "possible world," and its basic function in a formal language is to provide a locus of description for sets of hypothetical or counter factual assertions. This has been a tremendously powerful concept in formal linguistics for analyzing all types of sentences involving belief, wishing, hoping, imagining, etc.

In the formal description of commitments, the scenario is described as a series of facts true in a certain possible world. This world is then later 'actualized' or not according to the commitment being fulfilled or not. For additional background on possible worlds semantics, see Cresswell (1973) and Dowty (1978).

Possible worlds are added to the semantic model as the set W (of worlds). The sets in the model are therefore as follows:

$$M = \langle V,E,C,N,T,W,F \rangle.$$

where

V = truth values
E = entities
C = character strings
N = numbers
T = times
W = possible worlds
F = interpretation function.

SUMMARY

The focus of this paper has been to show how data management type databases can be interpreted in a formal, logical language. The motivation of this reformulation was to provide a formal basis for artificially intelligent managerial decision support systems. Specific advantages discussed were the capability to do logical inferencing on the contents of such databases and a framework for elaborating an explicit semantics for the symbols in the database.

REFERENCES

Anthony, R.B. 1965. Planning and Control Systems. Boston: Division of Research, Graduate School of Business Administration, Harvard University.

Codd, E.F. 1970. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM* 13(June):377-387.

Codd, E.F. 1971. A Data Base Sublanguage Founded on the Relational Calculus. Proceedings of ACM SIGFIDET Workshop on Data Description, Access, and Control, San Diego, California.

Cresswell, M.J. 1973. Logics and languages. London: Nethuen & Co. Ltd.

Date, C.J. 1977. An Introduction to Database Systems. Second Edition. Reading, Massachusetts: Addison-Wesley Publishing Co.

Dowty, D.R. 1978. A Guide to Montague's PTQ. Indiana: Indiana University Linguistics Club.

Kalish, D, R. Montague, and G. Mar. 1964 and 1980. Logic—Techniques of Formal Reasoning. New York: Harcourt Brace Jovanovich Inc.

Lee, R.M. 1980. CANDID—A Logical Calculus for Describing Financial Contracts. Philadelphia, Pennsylvania: Department of Decision Sciences, The Wharton School, University of Pennsylvania.

Lee, R.M. 1981a. A Formal Description of Contractual Commitment. WP-81-156. Laxenburg, Austria: International Institute for Applied Systems Analysis.

Lee, R.M. 1981b. CANDID Description of Commercial and Financial Concepts: A Formal Semantics Approach to Knowledge Representation. Forthcoming WP. Laxenburg, Austria: International Institute for Applied Systems Analysis.

Quillian, M.R. 1968. Semantic Memory. In Semantic Information Processing. pp. 227-268. Cambridge, Massachusetts: MIT Press.

Smith, J.M. and D.C.P. Smith. 1977. Database Abstractions: Aggregation and Generalization. ACM Transactions on Database Systems 2(June):105-133.

Strawson, P.F. 1963. Individuals—An Essay in Descriptive Metaphysics. Garden City, N.Y.: Doubleday and Co.

Suppes, P. 1957. Introduction to Logic. New York: Van Nostrand Co.

Tarski, A.N. 1931 and 1956. Logic, Semantics, Metamathematics. Oxford: Oxford University Press.

von Wright, G.H. 1968. An Essay in Deontic Logic and the General Theory of Action. Acta Philosophica Fennica, Fasc. XXI. Amsterdam: North Holland Publishing Co.