A Unifying Logic-based Formalism for Semantic Data Models

James Vorbach
St. John's University
Jamaica, New York

James Kowalski
University of Rhode Island
Kingston, Rhode Island

Abstract

Contrary to the Relational, Hierarchical, and Network Data Models which represent an application's semantics implicitly in a particular data structure, Semantic Data Models implement well-known data abstractions (i.e. aggregation, isa, generalization, specialization, association, and collection) in order to represent the semantics explicitly. A common formalism represents a framework in which data model formalisms can be transcribed and provides a uniformity useful for theoretical investigations. In this paper a first-order logic is introduced as a common formalism for analyzing Semantic Data Models. A schema is viewed as a first-order theory and the database as a model of this theory.

Relations in the logic correspond to the data abstractions mentioned and axioms are given which describe their properties. Every schema contains these 'general' axioms in addition to those specific to the real-world application. The presence of the 'general' axioms provides a logical environment in which schemas can be developed.

1.0 Introduction

Numerous investigations have been made in the data model area in order to seek more appropriate formalisms for accurately representing the real-world application. These investigations have resulted in a relatively new class of data models called semantic data models [CHEN76, SMITH77a, SMITH77b, CODD79, BRODIE81, HAMMER81, MCLEOD81, SHIPMAN81, HULL84, ABITEBOU87]. Semantic data models provide constructs for explicitly representing the semantics of the application. In contrast, the conventional models, i.e., relational, hierarchical, and network, implicitly represent the application semantics in their model data structures. The constructs in semantic data models implement information modeling tools called data abstractions. These abstractions enable a complex world to be examined in terms of a simplified world that incorporates the most significant points. Most importantly, data abstractions provide the basis for a stepwise design methodology for databases [BRODIE81, HAMMER81, MCLEOD81].

Each data model has its own structuring mechanism from which to build application schemas. In the conventional data models this mechanism is in terms of data structures. In the semantic data models this mechanism is in terms of semantic structures expressed in some textual language or in graph-theoretic terms. A common formalism is a unifying framework in which the structural component of various data models can be transcribed. A common formalism provides a uniform environment that enhances theoretical investigations of data models and their schemas. Specifically, such formalisms have been used to study the relative information capacity between schemas [HULL84], update propagation [ABITEBOU87], view integration, and automatic program conversion [JACOBS82]. A common formalism has also been used to provide a basis for a database language for heterogenous databases as well as an environment for developing application schemas [RYBINSKI87].

First-order predicate logic has several advantages as a representation language such as its expressive power, its proof theory, and its well-defined semantics provided by its foundation in set theory. In a database logic formalism, predicates correspond to the relationships between objects in the database. The schema is represented by formulas in the logic and the database represents the set-theoretic semantics.

Previously [JACOBS82, KUPER84, RYBINSKI87] logic formalisms were introduced which generalize the three conventional models. This paper introduces a logic formalism for analyzing semantic data models. Schemas in these models are expressed as first-order theories and the database is a model of this theory. Predicates and relations in the logic correspond to the data abstractions common in semantic data models. Axioms are given which specify properties of these abstractions. These axioms are referred to as 'general' axioms and are common to all theories/schemas in the logic.

The logic formalism provides a unifying theory for semantic data models and a logical environment in which these models and their conceptual schemas can be analyzed.

2.0 Data Abstractions

Abstraction is the process in which the essential details of a concept are emphasized and the irrelevant details with respect to a particular context are suppressed. Data abstraction is the application of this process to database design. Our focus here is on the data abstractions which provide the basis for a stepwise design methodology for databases [SMITH77a, SMITH77b, BRODIE81, HAMMER81, MCLEOD81]. Data abstractions exist that capture the semantics of relationships that are commonplace in the real world. These data abstractions then are used as primitives to represent complex database applications. In general, these data abstractions can be classified as forms of composite-component relationships and subtype-supertype relationships.

2.1 Composite - Component Relationships

A composite type represents the cartesian product of its components. Thus properties of a component type become properties of the composite type. For example, the 'Projects Assigned' type in Figure 1 is the composite of the 'Employee' and 'Project' types. Properties of these component types are properties of 'Projects Assigned' also. A 'Projects Assigned' object is characterized by an 'Emp_name', 'Proj_name', and the other properties of these component types. This upward inheritance of properties is characteristic of the data abstractions - *aggregation*, *association*, and *collection*.

The composite-component relationship, '**A** is a *component* of **B**', can have two interpretations. Firstly, **A** can be a *property* of **B** where a *property* is an atomic type. This relationship between a type and its properties is the *aggregation* abstraction and the composite type is called the *aggregate*. For example in Figure 1, the *aggregate* 'Employee' represents the composition of properties that describe each employee, i.e., 'Emp_name', 'Emp_no', and 'E_Address'. Secondly, **A** can be an aggregate component of **B**. This is called the *association* abstraction. The types 'Employee', 'Projects Assigned', and 'Project' comprise an *association* representing the relationship between employees and projects, i.e., the assignment of projects to employees. Previously [SMITH77a] these two interpretations were not distinguished. We do so here because it is useful when describing their formal properties in the logic.

The *collection* abstraction [HULL84] represents a restricted form of *association*, i.e., a *unary association*. Given two types **A** and **B**, '**A** is a *collection* of **B**' if instances of **A** are subsets of instances of **B**. For example, 'Class' is a collection of 'Student'. Each 'Class' instance contains a subset of the 'Student' instances. Properties for type 'Class', i.e. 'Section Number', express a group property for the set of 'Student' instances in each 'Class' instance.

2.2 Subtype - Supertype Relationships

The sets of instances from two types related in this manner are characterized by set inclusion. The instances of the subtype are a subset of the instances of the supertype. For example in Figure 1, 'Accountant' is a subtype of 'Employee', and consequently, 'Accountant' instances are also 'Employee' instances. Since all subtype intances are also supertype instances, properties of the supertype are also properties of the subtype. This downward inheritance of properties is characteristic of the data abstractions *isa*, *generalization*, and *specialization*.

'**A** *isa* **B**' means '**A** is a *subtype* of **B**'. In the figure 'Employee' is the supertype having the subtypes 'Engineer', 'Accountant', and 'Secretary'. The subtype instances inherit the properties of the supertype 'Employee', namely 'Emp_no', 'Emp_name', and 'E_Address'. Each subtype can be viewed as a restriction applied to the supertype, as for example, the set of 'Engineer' instances is that subset of 'Employee' instances who are engineers.

Generalization and *specialization* are *isa* abstractions having additional constraints. The constraints given here, which distinguish *generalization* and *specialization*, follow from those given by Abiteboul and Hull in the IFO Data Model [ABITEBOU87]. *Generalization* involves creating a supertype from a set of types. The types are generalized to construct the supertype whose properties are those shared by each type. Additional constraints which characterize *generalization* abstractions are *disjointness*, *union*, and *singular parent*. Each subtype's set of instances are *disjoint* from that of the other subtypes in the *generalization*. The *union* of the subtypes' instances equals the instances of the supertype. Lastly, a subtype can have only one supertype, i.e., can participate in only one subtype-supertype relationship.

Specialization represents the creation of subtypes of a particular type. Thus, unlike *generalization*, *specialization* is derived in a top-down manner. Each subtype inherits the properties of this type, augmenting the set of properties peculiar to its 'specialty'. For example, in Figure 1, 'Gov't Contracts' is a specialization of 'Project', representing that subset of projects that are contracted from the government. A constraint which characterizes the *specialization* abstraction is the *extension to a common parent* [ABITEBOU87]. A subtype may participate in more than one *specialization* but the supertype extension (i.e., the supertype of this supertype and so on) will lead to a common supertype of this *specialization* hierarchy (Figure 2).

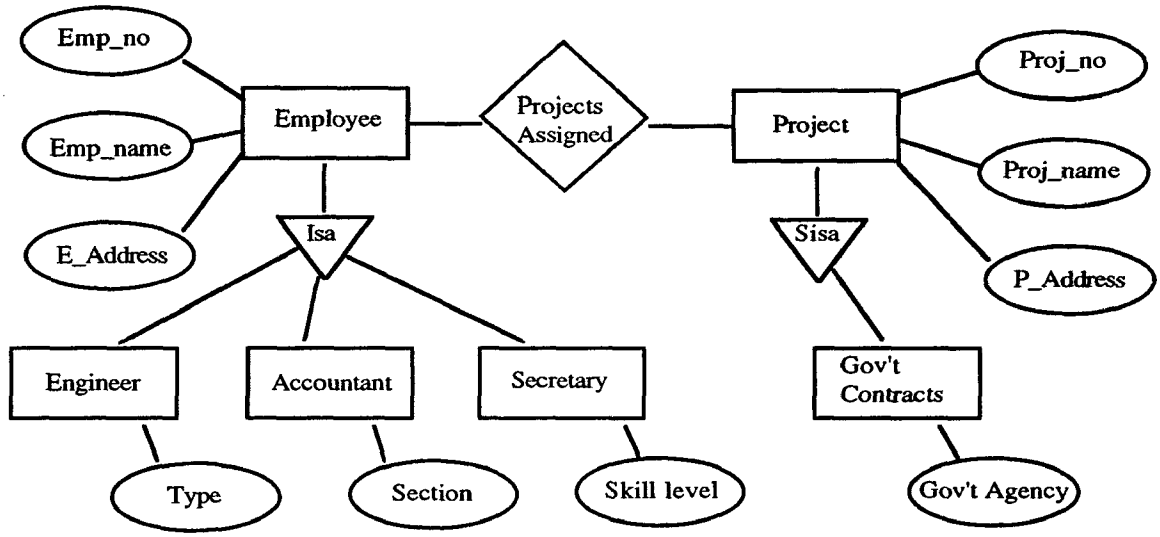

Figure 1. Employee - Project Schema

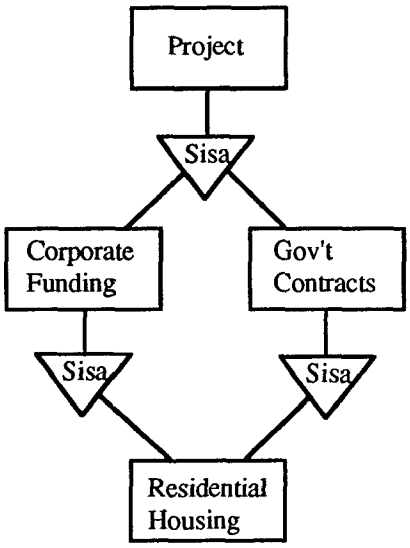

Figure 2. Extension to a Common Parent Constraint in Specialization Abstractions

3.0 Predicate Logic Formalisms

Consider the set of assertions in Figure 3(a). These assertions can be expressed as the first-order logic sentences in Figure 3(b).

1. John is an accountant.
2. Bill is an engineer.
3. Every accountant is an employee.
4. Every engineer is an employee.
5. Every employee has a salary.

(a)

1. ACCOUNTANT (John)
2. ENGINEER (Bill)
3. $\forall$ x (ACCOUNTANT (x) $\Rightarrow$ EMPLOYEE (x))
4. $\forall$ x (ENGINEER (x) $\Rightarrow$ EMPLOYEE (x))
5. $\forall$ x (EMPLOYEE (x) $\Rightarrow \exists$ y (SALARY (x, y)))

(b)

Figure 3

The *first-order language* in which this application is described consists of countable sets of *variables* and *constants*, the *predicates/relations* ACCOUNTANT, ENGINEER, EMPLOYEE, and SALARY, and the *logical operators* typical in first-order predicate logic $\Rightarrow$, &, $\vee$, $\forall$, $\exists$, ~. Atomic formulas in this language are of the form $P(s_1)$ or $P(s_1,s_2)$ where P is a predicate/relation and s_i is a variable or constant. Well-formed formulas (abbreviated wff's) are constructed from atomic formulas and, in addition, sequences of atomic formulas and operators.

A *first-order theory* consists of a *language*, a set of *logical axioms*, a set of *non-logical axioms*, and *inference rules*. *Axioms* are a closed wff's (wff's having no free variables) that describe various properties. The *logical axioms* are those common to every first-order predicate logic. *Non-logical axioms* express properties specific to an application such as those in Figure 3(b).

The *inference rules* provide a mechanism for deriving new wff's from the axioms in the theory. Derived wff's are called *theorems*. For example, a theorem in the previous example is EMPLOYEE(Bill) which would be derivable from axioms 2, 4 and the modus ponens inference rule (see 4.4).

4.0 The Logic Formalism

In this section a first-order theory is introduced which expresses axiomatically schemas in semantic data models. As previously discussed, data abstractions are useful tools for organizing application knowledge and consequently, are implemented by semantic data model formalisms. The predicates and relations in the logic correspond to the data abstractions that are supported in current models. Thus, the logic is at least as expressive as these models and provides for them a logic-theoretic semantics.

The first-order database logic, is formally defined to consist of the language in 4.1, the terms and well-formed formulas defined in 4.2, the set of logical and non-logical axioms in 4.3 and the inference rules in 4.4.

4.1 The Language

The language **L** is the alphabet of symbols from which formulas in the logic are constructed. Certain conventions are used to make clear the sorts of 'things' over which the variables range. Let **T** be the set of types t in the application domain. Let **P** be the set of properties p in the domain and let $\mathbf{P_t}$ represent the set of properties for a particular type t. The symbol x represents an object or an instance of a type. The symbol v represents a value or an instance of a property. This notation is discussed in detail in [VORBACH90] where the interpretation for the language is given.

The language **L** contains the following sets of denumerable constants and variables:
1) constants $\{c_1, c_2, c_3, \ldots\}$
2) objects $\{x_1, x_2, x_3, \ldots\}$;
3) objects of a particular type $t \in \mathbf{T}$, $\{x_1^t, x_2^t, x_3^t, \ldots\}$;
4) values $\{v_1, v_2, v_3, \ldots\}$;
5) values of particular properties $p \in \mathbf{P}$, $\{v_1^p, v_2^p, v_3^p, \ldots\}$;
6) variables for types in **T** $\{t_1, t_2, t_3, \ldots\}$;
7) variables for properties in **P** $\{p_1, p_2, p_3, \ldots\}$.

The language, **L**, contains of the following operators:
1) the logical connectives $\{\&, \vee, \sim, \Rightarrow\}$;
2) the universal quantifier, $\forall$, and the existential quantifier $\exists$.

The predicates and relations in **L** are classified into four categories. The *sorting predicates* are used to identify 'things' as being of a particular class or type. τ identifies objects of a particular type. θ identifies values of a particular property. α identifies members of the set **T**. κ identifies members of the set **P**. The *constraint predicates* name additional properties of particular data abstractions. The *abstraction relations* correspond to the data abstractions discussed. The *mapping relations* correspond to the functional relationships that are characteristic of aggregation and association. The notation used here for the mapping relations is a schema for those which occur in a particular applied theory.

The language, **L**, contains the following predicates and relations:
1) the sorting predicates τ, θ, α, and κ ;
2) the unary constraint predicates UNION, SING, DISJ and ESING;
3) the binary abstraction relations ISA, GISA, SISA, PART_OF, COL, and PROP;
4) the binary mapping relations
$\{\xi_{t_i}^{t_j} \mid t_i, t_j \in \mathbf{T}\}$ and $\{\gamma_{p_i}^{t_j} \mid p_i \in P_{t_j}, t_j \in \mathbf{T}\}$;
5) the equality relation = .

4.2 Definitions of Terms and Well-Formed Formulas

A *term* is a variable or constant symbol in **L**.

A *well-formed formula* (wff) is defined as follows:
1) if s_1 and s_2 are terms, then the expressions $\theta(s_1, s_2)$ and $\tau(s_1, s_2)$ are wff's;
2) if s_1 and s_2 are terms, $p_i \in P_{t_j}$, and $t_j \in \mathbf{T}$, then the expression $\gamma_{p_i}^{t_j}(s_1, s_2)$ is a wff;
3) if s_1 and s_2 are terms and $t_i, t_j \in \mathbf{T}$, then the expression $\xi_{t_i}^{t_j}(s_1, s_2)$ is a wff ;
4) if s is a term, then the expressions $\alpha(s)$, $\kappa(s)$, UNION(s), SING(s), DISJ(s) and ESING(s) are wff's;
5) if s_1 and s_2 are terms, then the expressions ISA(s_1, s_2), GISA(s_1, s_2), SISA(s_1, s_2), PART_OF(s_1, s_2), COL(s_1, s_2), and PROP(s_1, s_2) are wff's;
6) if A_1 and A_2 are wff's, then $\sim A_1$, $A_1 \vee A_2$, $A_1 \& A_2$, $A_1 \Rightarrow A_2$ are wff's;
7) if s is a term and A is a wff, then $\forall s$ A and $\exists s$ A are wff's;
8) no other expressions are wff's.

4.3 Logical and Non-Logical Axioms

Let $\mathbf{K}_{SDM}$ be the first-order theory utilizing the *language* **L** with the *logical axioms* typical in first-order predicate logic with identity [cf MENDELSO64], the *non-logical axioms* AX1 through AX14, and the *inference rules* in 4.4.

The non-logical axioms give properties specific to a theory's application. In the context of this work, a subset of the non-logical axioms (AX1 through AX14) are classified as 'general' axioms and describe properties of the data abstractions that apply in all applications. Consequently, any schema expressed as a first-order theory using this formalism will contain the following axioms within its set of non-logical axioms

AX1 through AX14 characterize ISA and PART_OF relations. D1 through D7 define 'higher-level' predicates/relations in terms of primitives.

AX1 Type arguments: $\forall t_1,t_2\ ISA(t_1,t_2) \Rightarrow \alpha(t_1) \ \&\ \alpha(t_2)$
This axiom restricts the argument variables in ISA. The α predicate corresponds to a type sort, restricting the variables to elements in the set of types **T** for the application.

AX2 Reflexivity: $\forall t_1\ ISA(t_1,t_1)$

AX3 Anti-Symmetry:
$\forall t_1,t_2\ (ISA(t_1,t_2) \ \&\ (t_1 \neq t_2) \Rightarrow \sim ISA(t_2,t_1))$
A subtype t_1 of another type t_2 cannot also be a supertype of t_2. Otherwise a redundancy would exist. In other words, if all instances of t_1 are instances of t_2 and all instances of t_2 are instances of t_1 then $t_1 = t_2$.

AX4 Transitivity:
$\forall t_1,t_2,t_3\ ((ISA(t_1,t_2) \ \&\ ISA(t_2,t_3)) \Rightarrow ISA(t_1,t_3))$

AX5 Incompatibility: $\forall t_1,t_2\ \sim(ISA(t_1,t_2) \ \&\ PART\_OF(t_1,t_2))$
This axiom specifies that a subtype t_1 of a supertype t_2 cannot also be a component t_1 of a composite type t_2.

AX6 Type Arguments:
$\forall p_1,t_1\ (PROP(p_1, t_1) \Rightarrow \kappa(p_1) \ \&\ \alpha(t_1))$
PROP identifies *p* as a property of *t*.The α predicate as previously noted is the type sort predicate and κ is the property sort predicate.

AX7 Downward Inheritance: $\forall t_1,t_2,p_1\ ((ISA(t_1,t_2) \ \&\ PROP(p_1, t_2)) \Rightarrow PROP(p_1, t_1))$

AX8 Type Arguments:
$\forall t_1,t_2\ (PART\_OF(t_1,t_2) \Rightarrow \alpha(t_1) \ \&\ \alpha(t_2))$

AX9 Anti-Reflexivity: $\forall t_1\ \sim PART\_OF(t_1,t_1)$

AX10 Anti-Symmetry: $\forall t_1,t_2\ (PART\_OF(t_1,t_2) \Rightarrow \sim PART\_OF(t_2,t_1))$

AX11 Transitivity:
$\forall t_1,t_2,t_3\ ((PART\_OF(t_1,t_2) \ \&\ PART\_OF(t_2,t_3)) \Rightarrow PART\_OF(t_1, t_3))$

AX12 Upward Inheritance:
$\forall t_1,t_2,p_1\ ((PART\_OF(t_1,t_2) \ \&\ PROP(p_1, t_1)) \Rightarrow PROP(p_1, t_2))$

AX13 Mixed Transitivity 1:
$\forall t_1,t_2,t_3\ ((PART\_OF(t_1,t_2) \ \&\ ISA(t_1,t_3)) \Rightarrow PART\_OF(t_3,t_2))$

AX14 Mixed Transitivity 2: $\forall t_1,t_2,t_3\ ((PART\_OF(t_1,t_2) \ \&\ ISA(t_3,t_2)) \Rightarrow PART\_OF(t_1,t_3))$

The notation $\equiv$ (i.e., $A_1 \equiv A_2$) in the following definitions is an abbreviation for $(A_1 \Rightarrow A_2) \ \&\ (A_2 \Rightarrow A_1)$. D1 through D4 define the constraint predicates SING, ESING, UNION, and DISJ respectively. D5 through D7 define the abstraction relations GISA, SISA, and COL respectively.

D1: $\forall t_1\ (SING(t_1) \equiv \forall t_2,t_3\ ((ISA(t_1,t_2) \ \&\ ISA(t_1,t_3)) \Rightarrow (t_2 = t_3 \vee ISA(t_2,t_3) \vee ISA(t_3,t_2))))$
This predicate corresponds to the *singular parent* constraint. The subtype t_1 is restricted to having only a single parent, unless the multiple parents are isa-related (i.e., one parent is the subtype of the other).

D2: $\forall t_1\ (ESING(t_1) \equiv \forall t_2,t_3\ ((ISA(t_1,t_2) \ \&\ ISA(t_1,t_3) \ \&\ t_2 \neq t_3) \Rightarrow \exists t_4\ (ISA(t_2,t_4) \ \&\ ISA(t_3,t_4) \ \&\ \forall t_5(ISA(t_2,t_5) \Rightarrow ISA(t_5,t_4)))))$
The ESING predicate corresponds to the *extension to a common parent* constraint which characterizes *specialization* abstractions. The variable t_4 represents the common parent of the *isa* abstractions that contain the subtype t_1. For all 'intermediate' types, t_5, in these *isa* abstractions, t_4 must be a common parent.

D3: $\forall t_1\ (UNION(t_1) \equiv \forall x_1(\tau(x_1,t_1) \Rightarrow \exists t_2(ISA(t_2,t_1) \ \&\ t_1 \neq t_2 \ \&\ \tau(x_1,t_2))))$
The UNION predicate corresponds to the *union* constraint. It constrains objects, x_1, of supertype t_1 to be objects of a subtype t_2 of t_1. From the ISA relation, the converse holds. Thus the supertype's objects are the union of its subtypes' objects.

D4: $\forall t_1\ (DISJ(t_1) \equiv \forall t_2,t_3\ ((ISA(t_2,t_1) \ \&\ ISA(t_3,t_1) \ \&\ \sim ISA(t_2,t_3) \ \&\ \sim ISA(t_3,t_2)) \Rightarrow \forall x_1(\tau(x_1,t_2) \Rightarrow \sim\tau(x_1,t_3))))$

The DISJ predicate corresponds to the *disjointness* constraint. It constrains the sets of objects of distinct subtypes (t_2 and t_3) of t_1, that are not isa-related, to be disjoint.

D5: $\forall t_1,t_2$ (GISA(t_1,t_2) ≡ (ISA(t_1,t_2) & UNION(t_2) & DISJ(t_2) & SING(t_1) & $t_1 \neq t_2$))
A *generalization* abstraction in a schema is represented in the formalism by axioms of the form GISA(t_1, t_2). An axiom exists for each subtype in the abstraction. The predicates UNION, SING, and DISJ correspond to the *union*, *singular parent*, and *disjointness* constraints which characterize *generalization* abstractions.

D6: $\forall t_1,t_2$ (SISA(t_1,t_2) ≡ (ISA(t_1,t_2) & ESING(t_1) & $t_1 \neq t_2$))
A *specialization* abstraction in a schema is represented in the formalism by axioms of the form SISA(t_1, t_2) and an axiom also exists for each subtype in the abstraction. The predicate ESING corresponds to the *extension to a common parent* constraint defined previously.

D7: $\forall t_1,t_2$ (COL(t_1,t_2) ≡ (PART_OF(t_1,t_2) & ~$\exists t_3$ (PART_OF(t_3,t_2) & $t_1 \neq t_3$ & ~ISA(t_1,t_3))))
A *collection* abstraction in a schema is represented in the formalism by an axiom of the form COL(t_1, t_2). The collection relation COL is defined in terms of the association relation PART_OF and the existential clause which represents the unary component restriction that characterizes collection abstractions.

4.4 Inference Rules

The inference rules of K_{SDM} are those typical of first-order predicate logic formal systems [cf MENDELSO64].

Generalization: Given A, assert $\forall x_i A$ where A is a wff and x_i is a variable term.
Modus Ponens: Given A_1 and $A_1 \Rightarrow A_2$, assert A_2 where A_1 and A_2 are wff's.

The following two examples illustrate how the formalism can express information contained in semantic data model schemas and databases. Example 4.1 represents the schema in Figure 1 and example 4.2 illustrates a database query

Example 4.1: A first-order theory for a specific database application contains the logical axioms common to first-order predicate logic and the non-logical axioms corresponding to the application. The non-logical axioms will contain the 'general' axioms specified in K_{SDM} as well as the axioms specific to a given schema. In Figure 4 the schema-specific, non-logical axioms are given as well as the mapping relations in the language representing the schema in Figure 1.

Example 4.2: The query 'What are the names of the employees who have employee numbers 1, 2, or 3 ?' is represented in the logic formalism by the wff in Figure 5. This query contains the free variable $v_1^{Emp\_name}$ and is interpreted as the set of employee names in a database that satisfy the query.

ISA (Accountant, Employee)
ISA (Engineer, Employee)
ISA (Secretary, Employee)
SISA (Gov't-Contracts, Project)

PART_OF (Employee, Projects Assigned)
PART_OF (Project, Projects Assigned)

PROP (Emp_no, Employee)
PROP (Emp_name, Employee)
PROP (E_Address, Employee)
PROP (Type, Engineer)
PROP (Section, Accountant)
PROP (Skill level, Secretary)
PROP (Gov't Agency, Gov't Contracts)
PROP (Proj_no, Project)
PROP (Proj_name, Project)
PROP (P_Address, Project)

$\xi_{Employee}^{Projects\ Assigned}$, $\xi_{Project}^{Projects\ Assigned}$

$\gamma_{Emp\_no}^{Employee}$, $\gamma_{Emp\_name}^{Employee}$, $\gamma_{E\_Address}^{Employee}$,
$\gamma_{Type}^{Engineer}$, $\gamma_{Section}^{Accountant}$, $\gamma_{Skill\ level}^{Secretary}$,
$\gamma_{Gov't\ Agency}^{Gov't\ Contracts}$, $\gamma_{Proj\_no}^{Project}$, $\gamma_{Proj\_name}^{Project}$,
$\gamma_{P\_Address}^{Project}$

Figure 4

5.0 Conclusion

In this paper a logic formalism was introduced to construct first-order theories representing semantic data model schemas. This formalism generalizes these models in the sense that data abstractions common in these models correspond to relations in the logic. The formalism then

provides a basis for theoretical investigations of semantic data models.

A designer using this formalism is presented with a logical environment with which to develop his or her schema in terms of the axioms given which specify properties of the data abstractions. Since the logic formalism is first-order, the standard theorems of first-order theories are applicable. Two specific database issues, i.e. schema consistency and relative information content, are examined in [VORBACH90] in light of this formalism.

$$\exists x_1^{Employee}((\gamma_{Emp\_no}^{Employee}(1, x_1^{Employee}) \,\&\, \gamma_{Emp\_name}^{Employee}(v_1^{Emp\_name}, x_1^{Employee})) \vee (\gamma_{Emp\_no}^{Employee}(2, x_1^{Employee}) \,\&\, \gamma_{Emp\_name}^{Employee}(v_1^{Emp\_name}, x_1^{Employee})) \vee (\gamma_{Emp\_no}^{Employee}(3, x_1^{Employee}) \,\&\, \gamma_{Emp\_name}^{Employee}(v_1^{Emp\_name}, x_1^{Employee})))$$

Figure 5

6.0 References

[ABITEBOU87] Abiteboul, S., Hull, R., "IFO: A formal semantic database model", *ACM Trans. on Database Systems*,12,4 (Dec. 1987) 525 - 565.

[BRODIE81] Brodie, M., "Data abstraction for designing database-intensive applications", *Proc. Workshop on Data Abstraction, Databases and Conceptual Modelling, June 23-26 1980, Pingree Park, Colorado*, 101-103.

[CHEN76] Chen, P., "The entity-relationship model - toward a unified view of data", *ACM Trans. on Database Systems*, 1,1 (March 1976) 9-36.

[CODD79] Codd, E., "Extending the database relational model to capture more meaning", *ACM Trans. on Database Systems*, 4,4 (Dec. 1979) 397-434.

[HAMMER81] Hammer, M., "Data abstractions for databases", *Proc. Workshop on Data Abstraction, Databases and Conceptual Modelling, June 23-26 1980, Pingree Park, Colorado*, 77-82.

[HULL84] Hull, R., Yap, C., "The format model: A theory of database organization", *Journal of the ACM*, 31, 3 (July 1984) 518-537.

[JACOBS82] Jacobs, B., "On database logic", *Journal of the ACM*, 29, 2 (Apr. 1982) 310-332.

[KUPER84] Kuper, G., Vardi, M., "A new approach to database logic", *Proc. 3rd ACM Symp. on Principles of Database Systems, Apr. 2-4, 1984, Waterloo, Ontario, Canada*, 86-96.

[MCLEOD81] McLeod, D., Smith, J., "Abstractions in databases", *Proc. Workshop on Data Abstraction, Databases and Conceptual Modelling, June 23-26 1980, Pingree Park, Colorado*, 19-25.

[MENDELSO64] Mendelson, E., "Introduction to Mathematical Logic," Van Nostrand, NY (1964).

[RYBINSKI87] Rybinski, H., "On first order logic databases", *ACM Trans. on Database Systems*, 12,3 (Sept. 1987) 325-349.

[SHIPMAN81] Shipman, D., "The functional data model and the data language DAPLEX", *ACM Trans. on Database Systems*, 6,1 (March 1981) 140-173.

[SMITH77a] Smith, D., Smith, J., "Database abstractions: aggregation", *Communications of the ACM*, 20,6 (June 1977) 405-413.

[SMITH77b] Smith, D., Smith, J.,"Database abstractions: aggregation and generalization", *ACM Trans. on Database Systems*, 2,2 (June 1977) 105-133.

[VORBACH90] Vorbach, J., "A unifying logic-based formalism for semantic data models", Ph.D Dissertatation, Dept of Computer Science and Statistics, University of Rhode Island, Kingston, RI (1990)