On the Semantic Equivalence of Heterogeneous Representations in Multimodel Multidatabase Systems

Dipayan Gangopadhyay
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598

Thierry Barsalou
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598

Abstract

We present in this paper an extensible metalevel system, called M(DM), in which the syntax and the semantics of data models, schemas, and databases can be uniformly described. We show with examples how to derive in M(DM) the semantic equivalence (or lack thereof) of symbols across different representation systems. We argue that, because of the inherent incompleteness of legacy databases, semantic equivalence must in general be ascertained by additional a posteriori assertions that are external to the representations under consideration.

1 Introduction

A database contains a symbolic representation of some real-world information. As with any other symbolic representation resulting from an abstraction process, a database captures only partial information about the real world. More fundamentally, all perceptual inputs are lost in the symbolic representation. From an interoperability perspective, the situation is further complicated because different databases may use different representation systems, such as different data models, different schema constructs, and different scales and units of measure for domain values. The essential problem of so-called "semantic heterogeneity" [10] is hence to determine a posteriori from *incomplete and heterogeneous representations* if symbols from multiple databases map to the same underlying reality.

To reconcile heterogeneous representations, we propose to adopt a Tarskian model of interpretation. That is, we define meaning functions which map symbols to elements in a common universe of discourse (yet another symbolic system). Two symbols are then equivalent if and only if they map to the same element in the common universe. We present a (second-order) logic-based universe of discourse, called M(DM), in which representation constructs from different data models, schemas and database extensions can be uniformly mapped. Hence, two database constructs are determined to be equivalent if and only if their M(DM) interpretations (collections of second-order formulae) are logically equivalent.

We handle incompleteness by defining external assertions about equivalence—that is, by augmenting the representations under consideration with additional facts. Along with other researchers [9], we argue that the use of additional assertions is the only viable solution in the context of legacy databases. Our justification comes from the writings of philosophers and logicians on the classical problem of *symbol grounding* [3, 6]. Following Charles Peirce, we recognize that the meaning of symbols is ultimately grounded in perceptual inputs. For example, one cannot tell for sure, purely by symbolic manipulation, if the names "Jones" and "Jones" in two databases refer to the same person. On the other hand, if perceptual inputs (e.g., pictures, fingerprints) accompany the symbols, we would be able to make a definite decision. In the case of legacy databases, which are devoid of such perceptual inputs, we may then have to adopt a new assertion of the form "Persons with the same last name and first name refer to the same human being." Elicitation of the semantic equivalence of symbols, however, is limited by the accuracy of these external, a posteriori assertions. Given such additional assertions, we show in this paper how the equivalence (or lack thereof) of different representations can be formally derived in the M(DM) framework.

2 M(DM): A Metalevel Framework

We briefly introduce here the salient aspects of M(DM). The reader is referred to [1] for a more complete exposition.

The basic underpinning of M(DM) is a type-based view of the universe. A data model is viewed as a collection of higher-order types, a database schema as a collection of first-order types instantiating the higher order types of its data model, and a database as a collection of instances of the types defined in its schema.

Each higher-order type (called metatype) in M(DM) formalizes the syntax and semantics of a data model construct by a collection of second-order logic formulae. These formulae express constraints simultaneously over the syntactic constituents of the data-model construct, its instances in the database schema and the database extensions of these instances. In the perspective of asserting semantic equivalence, we say

that *two types are equivalent if and only if the corresponding sets of second-order formulae are logically equivalent.*

Metatypes from different data models are further organized into an inheritance lattice where a metatype is a monotonic specialization of the metatype(s) higher up in the lattice. The reader may note that this organization leads to the extensibility of M(DM); that is, we can easily accommodate new data models by augmenting the metatype lattice [1]. We have indeed demonstrated M(DM)'s extensibility by capturing in a simple 15-metatype lattice a variety of data models including the relational model, the entity–relationship model, the structural model [12], DAPLEX [11], and Iris [5].

For the purpose of this paper, we introduce a subset of M(DM)'s language—M(DM)-L—as follows. A type $\mathcal{T}$ is defined by a type construction literal $\mathcal{T} \Leftarrow$ *Definition-Body*. For each such type definition, a corresponding set of *observer functions* and *observer predicates* becomes available. An M(DM)-L formula is a second-order logic formula over these type construction literals and observer functions and predicates.

For example, a metatype corresponding to the relation construct of the relational data model would be defined as follows: $\forall R : \mathcal{RELATION}(R) \leftarrow R \Leftarrow [a_j : D_j]$, where $1 \leq j \leq k$, $D_j \in \overline{DomainTypes}$ and $a_j \in \overline{Attributes}$. If r is a relational tuple of type R, we use the observer function $R.a_j$(r) to access the value of r's attributes. Moreover, the observer predicate R(r) indicates that r is of type R.

As another example, consider the relationship construct from the ER data model. A relationship is a k-ary association between entities. This fact is captured by the following definition of the metatype $\mathcal{RLSHIP}$: $\forall H : \mathcal{RLSHIP}(H) \leftarrow H \Leftarrow < E_1, \ldots, E_k >$, where $1 \leq j \leq k$, $k \geq 2$, $\mathcal{ENTITY}(E_j)$. That is, the relationship H is a k-ary association between k types, each of which is of metatype $\mathcal{ENTITY}$. Given this metatype definition, M(DM) provides an observer predicate $H(e_1, \ldots, e_k)$, which is true if the database instances e_1 of $E_1, \ldots, e_k$ of E_k have a k-ary association of type H.

The M(DM) metatypes, corresponding to the constructs of various data models, lay the foundation for several key steps in achieving interoperation of multimodel databases. M(DM) adheres to the federated approach. An M(DM) federation consists of multiple local databases together with their export schemas, which represent the sharable portions of these databases. Each export schema gets automatically mapped to an equivalent set of M(DM) types that, in turn, forms a set of M(DM)-L formulae; this set hence specifies an M(DM) import schema. The collection of import schemas for all local databases in the federation defines an M(DM) multidatabase schema. This process results in *DBMS independence*; multiple databases can be manipulated and queried with one language (i.e., M(DM)-L). Moreover, if an application at one site wants to view the entire federation according to its own "perspective," an appropriate view can be defined in M(DM)-L, thereby achieving *database transparency.*

The definition of views is consistent with M(DM)'s handling of types; a view V is a type that instantiates a valid metatype $\mathcal{M}$. Accordingly, the view definition must provide a type construction literal, observer functions and observer predicates. The view-definition syntax in M(DM)-L is then:

```
M(V), ( V ⇐ Definition-Body,
        Observer-Predicates,
        Observer-Functions )
      if   Computation-Body
```

Because a view is virtual, the definition expression has to be a conditional rule. The rule body, `Computation-Body`, provides the necessary specification to dynamically materialize the view in terms of the underlying database(s). In other words, whenever the logical formulae in `Computation-Body` are true, a view V is defined with the appropriate syntax. Moreover, V's observer predicates and functions derive their values from the appropriate values of the base-type observer predicates and functions.

As we shall see in Section 3, M(DM)'s view mechanism is of paramount importance for reconciling heterogeneous data representations and providing a uniform perspective on disparate databases.

3 Dealing with Semantic Heterogeneity

In this section, we present four examples of data heterogeneity. We illustrate how the mapping of various representations to M(DM) allows us to reason about the semantic equivalences of symbols. In addition, we show explicitly the critical role played by external assertions to compensate for the lack of completeness of legacy databases.

3.1 Discrepancies in data definition

The meal-cost problem is a simple example of semantic heterogeneity. Let us assume that two relational databases D_1 and D_2 have joined an M(DM) multidatabase federation. Both databases contain a relation called RESTAURANT with the following schema: RESTAURANT: Name: STRING, Address: STRING, AvgCost: FLOAT. The AvgCost attribute represents the average cost of a meal at a given restaurant. As briefly described in Section 2, the mapping of these two schemas into an M(DM) multidatabase schema produces a set of first-order type definitions.

At first glance (that is, by comparing D_1 and D_2 type definitions for RESTAURANT), it would appear that the two AvgCost attributes are semantically equivalent and could hence be used for such things as join operations across databases. This is not the case, however, because the interpretation of AvgCost happens to be different in the two databases. D_1.RESTAURANT.AvgCost corresponds to the base price, whereas D_2.RESTAURANT.AvgCost includes the

base price, tax, and gratuity. To reflect this new information, we assert a new equivalence between the AvgCost attributes in the two databases in the form of an M(DM)-L well-formed formula:

$$\forall r, D_1.\text{RESTAURANT}(r),\ D_2.\text{RESTAURANT}(r) \rightarrow D_2.\text{RESTAURANT.AvgCost}(r) = D_1.\text{RESTAURANT.AvgCost}(r) + f(D_1.\text{RESTAURANT.AvgCost}(r)),\ f(D_1.\text{RESTAURANT.AvgCost}(r)) > 0.$$

This additional assertion represents explicitly the fact that the average meal cost of a restaurant will be different in the two databases. As a result, we have augmented the M(DM) type definition of D_2.RESTAURANT to capture this basic definitional discrepancy.

As this example shows, discovering the existence of the function f is critical to proper interoperation; f's definition, however, is unlikely to be found in the input schema definitions. If f could be computed or derived in some way (for example, by using the local standard gratuity and sales tax), we could determine with certainty the equality or inequality of meal costs in the two databases. On the other hand, if f cannot be obtained, the potential use of AvgCost across D_1 and D_2 becomes very limited. We know that \$100 in D_1 is not equal to \$100 in D_2, but, for example, we cannot tell if a \$100 meal in D_1 is equivalent to a \$120 meal in D_2. In other words, M(DM), as any other Tarskian system, can address issues of representational heterogeneity (as shown in the remaining three examples) but, cannot, by itself, deal fully with the more general problems of incompleteness and symbol grounding.[1]

3.2 Differences in data structures: Single-model case

In this frequently encountered problem, the same information is represented with different data structures in two databases. Using the example of [7], suppose that, in a federation of stock-price databases, a relational database D_1 has a relation STOCKS with the following schema: STOCKS: Company: STRING, Day: DATE, Price: FLOAT, whereas a second relational database, D_2, has one relation per company as follows: COMPANY1: Day: DATE, DayPrice: FLOAT; COMPANY2: Day: DATE, DayPrice: FLOAT; and so on.

Clearly, although the two databases contain the same information, the type definitions in M(DM) are not equal. We can solve this problem, however, by defining a view on D_2 and asserting simultaneously equivalence of D_2 and the view, and of the view and D_1. (This approach is conceptually similar to the notion of *transformational equivalence* described in [2].) D_1.STOCKS is equivalent to the set D_2.COMPANY1, ..., D_2.COMPANYn if D_1.STOCKS can be derived from the set D_2.COMPANY1, ..., D_2.COMPANYn by an information-preserving (isomorphic) transformation.

[1] Accordingly, we will assume in the remainder of this paper that two symbols, which are syntactically identical, do refer to the same real-world entity.

In this case, such a transformation is a higher-order view (that is, a view whose definition depends on the data) over D_2 according to D_1's schema for stocks. Following the view-definition syntax introduced in Section 2, the M(DM)-L expression for specifying such a view, VIEW.STOCKS, is:

```
RELATION(VIEW.STOCKS), ( VIEW.STOCKS ⇐
  [ Company:STRING, Day:DATE, Price:FLOAT ],
  VIEW.STOCKS(g), VIEW.STOCKS.Day(g) = n,
  VIEW.STOCKS.Company(g) = c,
  VIEW.STOCKS.Price(g) = f )
  if RELATION(D2.c), D2.c ⇐
       [ Day: DATE, DayPrice: FLOAT ],
    D2.c(x), D2.c.Day(x) = n,
    D2.c.DayPrice(x) = f.
```

To give the view its higher-order nature, the variable c assumes two different roles (attribute value and relation label) in the head and the body of the rule. We further derive the values of VIEW.STOCKS's two other observer functions (Day and Price) from the base data through two additional shared variables, n and f.

We now have two M(DM) types, D_1.STOCKS and VIEW.STOCKS, which are equivalent; that is, they are defined by identical sets of second-order logic formulae. It should be also clear that the view computation performs only simple reformatting operations and thus preserves all data contained in D_2's company relations. We can hence assert equivalence between D_2's set of company relations and VIEW.STOCKS. Note that, if the view definition had projected out some attributes or used an aggregation function (that is, information is lost from the base to the virtual representation), M(DM) would not accept this assertion.

As a result, because D_1 and VIEW.STOCKS, and VIEW.STOCKS and D_2 are equivalent, it follows that the representations in D_1 and D_2 are also semantically equivalent. Note finally that the same M(DM) view mechanism can be used for resolving other cases of structural discrepancies such as those caused by differing naming conventions and storage formats.

3.3 Differences in data structures: Multi-model case

The two previous examples assumed a common data model for the participating databases. We now show that M(DM) is flexible enough to deal with representational heterogeneities arising from the use of different data models at different sites. Indeed, our primary motivation in this work has been to specify an extensible framework for multimodel multidatabase systems [1].

As an example, we explore the case where different data-model constructs are used to represent similar information. The problem is hence to assert semantic equivalence, or lack thereof, of these heterogeneous constructs.

Consider the integration of two naval databases, D_1 and D_2, that both represent information about SHIPS and their PORT-OF-REGISTRATION. D_1 is a relational database with the following schema: SHIP: ShipId:

INTEGER, ShipName: STRING, Port-of-Registration: STRING. D_2, on the other hand, is an entity-relationship (ER) database, which defines PORT-OF-REGISTRATION as a relationship type from the entity type SHIP (ShipId: INTEGER, ShipName: STRING) to another entity type CITY (Name: STRING). Is D_1's relational attribute Port-of-Registration semantically equivalent to D_2's ER relationship type PORT-OF-REGISTRATION?

M(DM) allows us to move to a more abstract level where the meanings of relational attributes and ER relationships can be defined and manipulated. According to the M(DM) metatype definitions, each relational attribute defines a function,[2] and each ER relationship defines a logic predicate. More specifically, D_1's Port-of-Registration defines a function D_1.SHIP.Port-of-Registration such that, for each ship x, there is a unique string p such that D_1.SHIP.Port-of-Registration(x) = p; D_2's PORT-OF-REGISTRATION defines a binary predicate D_2.PORT-OF-REGISTRATION(x, y), which is potentially true for many ship–city pairs (x, y). In other words, D_2's predicate essentially defines a many-to-many association, whereas D_1's function strictly embodies a many-to-one association. M(DM) hence distinguishes the two representations on the basis of cardinality.

If, on the other hand, the relationship in D_2 is constrained to have a one-to-one cardinality, we can assert equivalence of the two representations using the approach demonstrated in Section 3.2. We first define a view on D_2 and subsequently derive equivalence of the two representations through this view. The view definition illustrates well our ability to mix metatype predicates (such as $\mathcal{RELATION}$, $\mathcal{ENTITY}$ and $\mathcal{RLSHIP}$) in a single M(DM)-L expression. (This capability enables M(DM) to uniformly support diverse data models in a multidatabase federation.) The view, VIEW.SHIP, is expressed in M(DM)-L as follows:

```
RELATION(VIEW.SHIP), ( VIEW.SHIP ⇐
  [ ShipId: INTEGER, ShipName: STRING,
    Port-of-Registration: STRING ],
  VIEW.SHIP(s), VIEW.SHIP.ShipId(s) = i,
  VIEW.SHIP.ShipName(s) = n,
  VIEW.SHIP.Port-of-Registration(s) = p )
  if ENTITY(D2.SHIP), D2.SHIP ⇐
       [ ShipId:INTEGER, ShipName:STRING ],
     ENTITY(D2.CITY),
     D2.CITY ⇐ [ Name: STRING ],
     RLSHIP(D2.PORT-OF-REGISTRATION),
     D2.PORT-OF-REGISTRATION ⇐
       < D2.SHIP, D2.CITY >,
     D2.SHIP(x), D2.CITY(y),
     D2.PORT-OF-REGISTRATION(x,y),
     D2.SHIP.ShipId(x) = i,
     D2.SHIP.ShipName(x) = n,
     D2.CITY.Name(y) = p.
```

For each ship x in D_2, there is now a unique city y such that D_2.PORT-OF-REGISTRATION(x, y); moreover, each city has a unique name as specified by the function D_2.CITY.Name(y) = p. Clearly, D_1.SHIP and VIEW.SHIP represent the port of registration as a unique association between a ship and a string. Using again the assertion of transformational equivalence between D_2's schema and VIEW.SHIP, we can hence reconcile D_1's relational attribute and D_2's ER relationship.

To conclude this example, note that, if D_2's entity type CITY had additional attributes (such as population and country), we would not be able to reconcile the two representations. In such a case, PORT-OF-REGISTRATION could not be reduced (or coerced) from an association between a ship and a city to an association between a ship and a string. D_2's schema and VIEW.SHIP would thus not be deemed equivalent by M(DM).

3.4 Key equivalence

The fundamental problem of key equivalence was recently stated by Pu as follows [8]. Given two keys in different local databases, we must decide if they refer to the same real-world entity. Because of the intrinsic incompleteness in legacy databases, no single algorithm can solve the general key-equivalence problem. Pu proposed a probabilistic algorithm for the domain of people's names. We tackle here a different but complementary aspect of the key-equivalence problem. In both cases, however, the central notion is that additional assertions are needed to elicit equivalence.

Let us consider two employee databases which contain different information on the same set of employees. An object-oriented database, D_1, has a class EMPLOYEE with three attributes SSNum: INTEGER, Salary: INTEGER, and Dpt: DEPARTMENT; each D_1 object is identified by a unique object identifier (OID). A relational database, D_2, has a relation PERSON with the following schema: PERSON: SS#: INTEGER, Name: STRING, DoB: DATE. We are interested in joining data from the two databases. For example, we may want to retrieve the name and salary of all employees born in 1960. The problem is hence to reconcile D_1's OID-based and D_2's value-based identification schemes.

Again, we declare that two entries in the two databases represent the same real-world employee by introducing external assertions. In this case, since both databases have an attribute for social-security numbers, we write the following statement:

$$\forall\, o \in D_1.\text{EMPLOYEE},\ \forall\, t \in D_2.\text{PERSON},\\ o \equiv t \Leftrightarrow D_1.\text{EMPLOYEE.SSNum}(o) = D_2.\text{PERSON.SS\#}(t).$$

Because SS# is the key for D_2.PERSON, however, M(DM) can only accept this assertion if a uniqueness constraint is also present on D_1.EMPLOYEE's SSNum attribute.[3] If such a constraint cannot be enforced (or if the social-security number was *not* in one of the two databases), other methods, such as Pu's matching algorithm on names, would have to be used to determine

[2] Function is used here in the mathematical sense. For any input x, there is a unique y such that $f(x) = y$.

[3] The notion of key uniqueness is part of M(DM)'s metatype definition for relations.

key equivalence. Typically, however, the certainty of two matching entries in D_1 and D_2 actually referring to the same physical person would be lost.

4 Conclusion

We described in this paper a concrete framework for understanding semantic heterogeneity in multidatabase systems. By casting the issue as one of symbolic representation, we identified two subproblems—*representational heterogeneity*, stemming from the use of different systems of representation, and *symbol grounding*, arising from the inherent incompleteness of legacy databases.

To address the question of representational heterogeneity, we presented M(DM)—an extensible, type-based framework in which different representations can be uniformly manipulated. In particular, we sketched the use of M(DM) for reasoning with constructs and structures from a variety of data models. Within such a framework, we can reduce the problem of semantic equivalence across heterogeneous representations to that of type equivalence, which in turn boils down to the assertion of logical equivalence of sets of second-order logic formulae. As we demonstrated in Sections 3.2 and 3.3, M(DM)'s powerful view mechanism plays a central role in this process.

To deal with the problem of symbol grounding, we advocated the use of external assertions for deriving the semantic equivalence of symbols. These assertions are expressed as M(DM)-L formulae, which can be incorporated into M(DM) multidatabase views. As Section 3.1 showed, however, external assertions are necessary but often not sufficient to fully make up for the incompleteness and inconsistencies of local databases. Elicitation of symbol equivalence is therefore limited by the precision of these external, a posteriori assertions. We believe nonetheless that this method of supplementing database schema information with additional statements is the only viable approach in the case of legacy databases.

In practical terms, we envision a system in which new external assertions are first checked for consistency with respect to the existing assertions (i.e., the M(DM)-L sentences derived from the mappings of data models and database schemas). If no inconsistency is found, the new assertions are added to the M(DM) multidatabase schema. Thereafter, queries about semantic equivalence of terms can be answered by the underlying inference machinery of M(DM). We are currently studying an implementation of M(DM) in OOLP [4], a language that embodies the necessary facilities of object orientation, metaprogramming, and logic programming.

References

[1] T. Barsalou and D. Gangopadhyay. M(DM): An open framework for interoperation of multimodel multidatabase systems. Technical Report RC 16965, IBM Thomas J. Watson Research Center, Yorktown Heights, NY. To appear in *Proceedings of the 8th International Conference on Data Engineering*, Phoenix, AZ, February 1992. IEEE.

[2] C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.

[3] J. Buchler, editor. *Philosophical writings of Peirce*. Dover, 1965.

[4] M. Dalal and D. Gangopadhyay. OOLP: A translation approach to object-oriented logic programming. In W. Kim, J.M. Nicolas, and S. Nishio, editors, *Proceedings of the First International Conference on Deductive and Object-Oriented Databases*, Kyoto, Japan, December 1989. Elsevier Science Publishers.

[5] D.H. Fishman, D. Beech, H.P. Cate, and E.C. Chow. Iris: An object-oriented database management system. *ACM Trans. on Office Information Systems*, 5(1):48–69, 1987.

[6] S. Harnad, editor. *Categorical perception: The groundwork of Cognition*. Cambridge University Press, 1987.

[7] R. Krishnamurthy, W. Litwin, and W. Kent. Language features for interoperability of databases with schematic discrepancies. In *Proceedings of the International Conference on Management of Data*, pages 40–49, Denver, CO, 1991. ACM SIGMOD.

[8] C. Pu. Key equivalence in heterogeneous databases. In *Proceedings of the First International Workshop on Interoperability in Multidatabase Systems*, pages 314–316, Kyoto, Japan, 1991. IEEE Computer Society.

[9] A.P. Sheth and S.K. Gala. Attribute relationships: An impediment in automating schema integration. Workshop on Heterogeneous Database Systems, December 1991.

[10] A.P. Sheth and J.A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[11] D. Shipman. The functional data model and the data language DAPLEX. *ACM Trans. on Database Systems*, 6(1):140–173, 1981.

[12] G. Wiederhold and R. ElMasri. The structural model for database design. In *Entity-Relationship Approach to System Analysis and Design*, pages 237–257. North-Holland, Amsterdam, The Netherlands, 1980.