A Rule-based Language for Deductive Object-Oriented Databases

A. M. Alashqur S.Y.W. Su H. Lam

Database Systems Research and Development Center
Computer and Information Sciences Department
University of Florida, Gainesville, FL 32611

ABSTRACT

The object-oriented view of an application world can be represented in the form of a network of classes and associations, which can be aggregation or generalization associations. At the extensional level, objects of different classes can be related (associated) with each other forming patterns of object associations. In this paper, we present a deductive rule-based language for Object-oriented databases. A deductive rule in this language derives new patterns of associations among objects of some selected classes if these objects fall in certain "base" or other derived patterns. The patterns of object associations derived by a rule are held in a subdatabase whose intension consists of some selected classes and their associations. In other words, the structure of a derived subdatabase is represented using the structural constructs provided by the object-oriented data model and hence can be uniformly operated on by other rules to further derive new subdatabases. Therefore, the world of subdatabases is closed under this rule-based language.

1. Introduction

Merging expert systems (ES) and database management systems (DBMS) technologies has drawn much interest in recent years [GAL84, ULL85, STO87, RAS88, MAI88]. This interest is motivated mainly by the need for future ESs that deal with large amounts of data as well as the need for future DBMSs that have deduction capabilities and, therefore, can support many of the new database application areas such as CAD/CAM, office automation, and multi-media databases. Several efforts have been made to design and integrate a deductive PROLOG-based rule language with a relational DBMS [JAR84, CHA84, VAS84, ULL85, CER86, STO87, MAI88, DEL88]. In this approach, deductive rules are declared in the form of a logic program against base relations in the database. Each rule defines a virtual relation that is derived from other base and/or virtual relations.

The integration of such PROLOG-based languages with relational databases is facilitated mainly by the fact that a relational database is **closed** under PROLOG-like deductive rules in the sense that the input to a deductive rule (i.e., the relations that the rule operates on) can be one or more relations and its output is always a relation. In other words, the output of a rule still belongs to the world of relations (i.e., it is modeled using the relational data model) and therefore can be uniformly operated on by other deductive rules to further derive new relations and so on. It is our belief that a deductive rule-based language that is designed for any data model has to preserve the **closure property** with respect to that data model meaning that the derived data must be structured and modeled using the same data model with which the "base" data are modeled.

Along a different research line, the database technology itself has been actively moving towards the Object-oriented (OO) approach, which is more appropriate than the relational approach for supporting the new database application areas mentioned above [HAM81, BAT85, BAN87, FIS87, HUL87, FOR88, SU89]. Semantic and OO data models can capture much more of the semantics of these application domains in a "natural" way. The term "OO data model" is used in this paper to refer to a data model that is structurally and/or behaviorally object-oriented [DIT86]. A structurally OO data model is one that encompasses at least the following characteristics: (1) It allows for defining aggregation hierarchies, (2) It allows for defining generalization hierarchies, and (3) It supports the unique identification of objects, that is, each object is assumed to have a unique object identifier (OID).

The OO view of an application world is represented in the form of a network of classes and associations, which can be aggregation or generalization associations. Object classes can be either **primitive** classes whose instances are of simple data types (e.g., integer, string, real) or **non-primitive** classes whose instances represent real world objects (e.g., Part, Employee). At the extensional level, instances of different classes can be related (associated) with each other forming patterns of object associations. In a previous work [ALA89a], we introduced the OO query language OQL, which is a high-level, non-procedural, and set-oriented language that allows the users to query the database by specifying the desired patterns of object associations in their queries. A behaviorally object-oriented data model, on the other hand, is one in which operations that describe the behavior of the objects of a class can be defined and registered with that class.

For a database system to be more appropriate for supporting the new database application areas cited above, the two, so far independent, research lines (i.e., deductive databases and OO databases) need to be merged leading to deductive OO databases. This can best be done by designing a deductive rule-based language that preserves the closure property with respect to OO data models (i.e., models that support aggregation, generalization, and the unique identification of objects). In this case, the output of a rule must be represented using the constructs of the OO data model in order to make it possible for such an output to be operated on uniformly by other rules to further produce other derived data and thus form inference chains.

In this paper, we present the design and implementation techniques of a deductive rule-based language for **OO** databases. A deductive rule in this language derives new patterns of associations among objects of some selected classes if these objects fall in certain "base" or other derived patterns. The patterns of object associations derived by a rule are held in a **subdatabase** (see Sections 3 and 4) whose intension (i.e., structure) consists of some selected classes and their associations. In other words, the structure of a derived subdatabase is represented using the structural constructs provided by an **OO** data model and hence can be uniformly operated on by other rules to further derive new subdatabases. Therefore, the world of subdatabases is closed under this rule-based language. To our knowledge, work on deductive rule-based languages of this kind for **OO** databases has not been reported in the literature.

This paper is organized as follows. After this introduction, we briefly describe the **OO** view of a University application domain, which will be used in later sections for defining some example deductive rules. In Section 3, we briefly describe the notion of a subdatabase and the **OO** query language OQL. Some of the constructs of OQL are used by the deductive rule-based language to be introduced in Section 4. In Section 5, we describe the transitive closure operation as can be performed in our language. The control strategies used for evaluating deductive rules are described in Section 6. Some concluding remarks are given in Section 7.

2. The Object-oriented View of Databases

We shall describe first the **OO** view of a university database as modeled by the **OO** semantic association model OSAM* [SU89]. The university schema shown in Figure 2.1 is then used in the remainder of this paper as the application domain for which example deductive rules are given. The concepts introduced in this paper are applicable to any **OO** data model and not limited to the OSAM* model.

A database schema is represented in OSAM* as a network of associated (inter-related) object classes. Graphically, object classes are represented as nodes and associations among object classes are represented as links. The resulting diagram is called the Semantic Diagram or **S-diagram**. In OSAM*, there are two types of object classes: Entity object classes **(E-class)** and Domain object classes **(D-class)** which are represented in the schema of Figure 2.1 as rectangular and circular nodes, respectively. The sole function of a D-class is to form a domain of values of a simple data type (e.g., integers, strings, etc.) from which descriptive attributes of objects draw their values. An E-class, on the other hand, forms a domain of objects which occur in an application's world (e.g., Faculty, Department, etc.). Each object of an E-class is represented by a system-generated unique object identifier (OID).

There are five types of links (associations) in OSAM*. Two of these association types appear in Figure 2.1, namely, Aggregation (A) and Generalization (G), which are also recognized in several other semantic and **OO** data models. A class can have several types of links and more than one link of each type emanating from it. In the S-diagram, links of the same type that emanate from a class are grouped together and labeled by the letter that denotes the association type.

As an example, in Figure 2.1, the E-class Person has two types of links: Aggregation links connecting Person to the D-classes SS# and Name and Generalization links to the E-classes Student and Teacher (i.e., Student and Teacher are subclasses of the superclass Person). An aggregation link represents an attribute and has the same name as the class it connects to unless specified otherwise (e.g., the link labeled Major which emanates from the class Student has a different name from the class it connects to). Aggregation links that emanate from an E-class and connect to D-classes are referred to as the descriptive attributes of that class (e.g., the attribute section# of the class Section). A class inherits all the aggregation associations that connect to or emanate from its superclasses. Figure 2.2 shows the actual view of the class Research Assistant (RA) in which all the associations inherited by RA from its super classes are explicitly represented. A detailed description of the OSAM* model can be found in [SU89].

3. Subdatabases and the OO Query Language OQL

In this section we first give the definition of a subdatabase as presented in [ALA89a] (in Section 4, we extend this definition by introducing the induced generalization association construct which enables a deductive rule to operate on a set of subdatabases rather than on a single subdatabase). Next, we briefly describe the object-oriented query language OQL. The motivation behind presenting OQL here is that some of the OQL constructs will also be used as part of the deductive rule-based language to be introduced in Section 4 and an example OQL query against the derived data will be given in that section.

3.1 Subdatabases

A subdatabase is a portion of the original database and consists of two parts: an **intensional association pattern** and a set of **extensional association patterns**. The intensional association pattern of a subdatabase is represented as a network of E-classes and their associations. For example, Figure 3.1 shows a certain subdatabase SDB of the original database of Figure 2.1. Figure 3.1a represents the intensional association pattern of this subdatabase, which consists of the classes Teacher, Section, and Course and their associations.

An extensional association pattern is a network of instances and their associations that belong to the classes and association types of the intensional association pattern. The set of extensional patterns of a subdatabase can be represented in the form of an **extensional diagram**. Figure 3.1b shows a possible extensional diagram for the subdatabase SDB, where the t's, s's, and c's are the OIDs for objects from the classes Teacher, Section, and Course, respectively. The interconnection of t3 and s4 in the figure is an example of an extensional pattern, which records the fact that object t3 is associated with object s4 (Teacher t3 is teaching Section s4).

In addition to the graphical representation, an extensional pattern can be represented as a tuple of OIDs. For example, <t1,s2,c1>, <c3> and <s5,c4> are some of the extensional patterns that appear in the subdatabase SDB (Figure 3.1b). The two extensional patterns <t2,s3,c1> and <t2,s3,c2> contain the same Teacher and Section instances but different

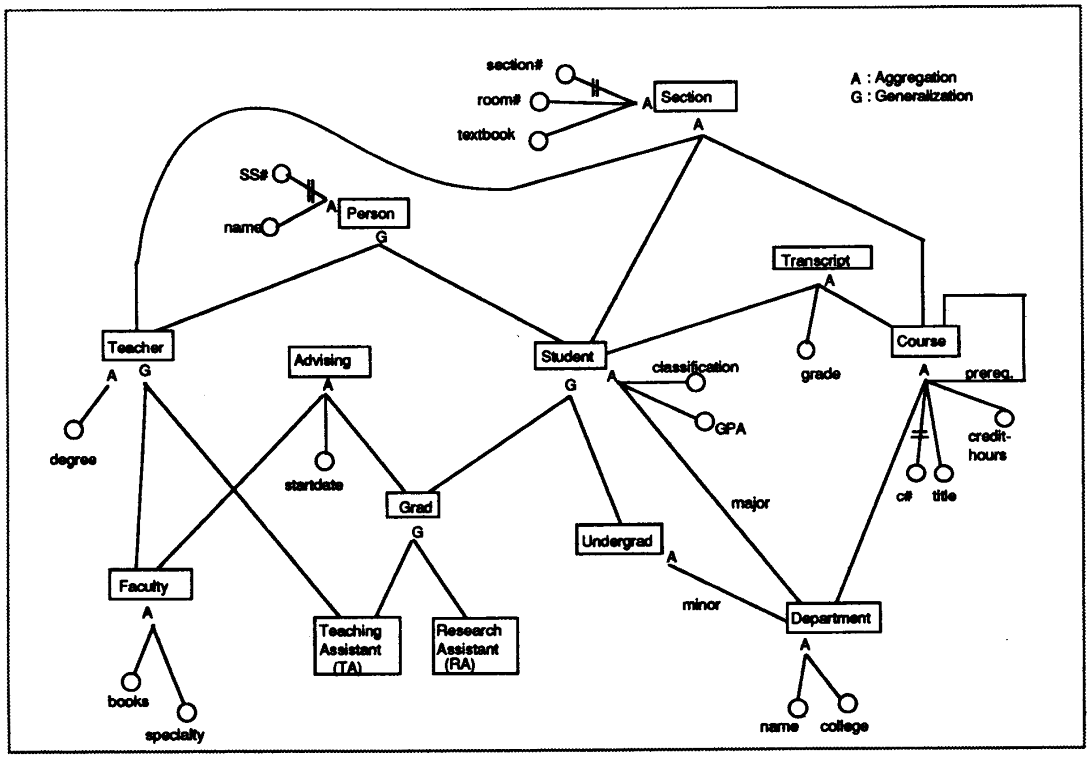

Figure 2.1: University Schema

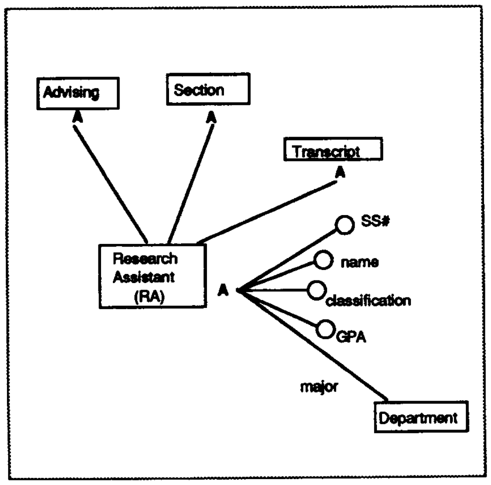

Figure 2.2: Class RA with all the Inherited Associations Explicitly Represented

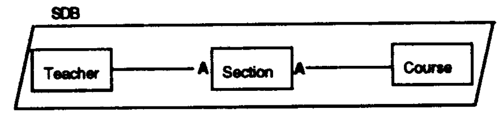

Figure 3.1a: The Intensional Pattern of a Subdatabase SDB

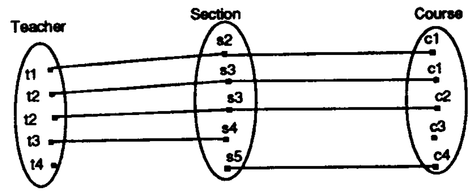

Figure 3.1b: A Possible Extensional Diagram for the Subdatabase SDB

Figure 3.1 The Intensional and set of Extensional Patterns of a Subdatabase

Course instances[1]. In the definition of a subdatabase as given in [ALA89a] it was assumed that all the descriptive attributes of a class that appears in a subdatabase also appear with it, by default, in the subdatabase. However, the descriptive attributes are not shown in Figure 3.1a to keep the figure simple.

An **extensional pattern type** is defined as the common template that is shared by several extensional patterns in a subdatabase. A pattern type is denoted by a tuple of class names. For example, <Teacher, Section, Course> is one of the extensional pattern types that exist in Figure 3.1b, which has as instances all the extensional patterns that contain Teacher, Section, and Course objects, i.e., the extensional patterns <t1,s2,c1>, <t2,s3,c1>, and <t2,s3,c2>. On the other hand, the extensional pattern <t3,s4> whose Course-component is Null (since the pattern does not contain any Course object) is of the type <Teacher,Section>. The five extensional pattern types present in the extensional diagram of Figure 3.1b are <Teacher,Section, Course>, <Teacher,Section>, <Section,Course>, <Teacher>, and <Course>.

3.2 The Object-oriented Query Language OQL

The philosophy underlying OQL is to allow the user to specify, first, the desired subdatabase by specifying its intensional pattern and the set of extensional pattern types that are of interest and then the operation(s) to be performed on the classes of the subdatabase. The search engine of the underlying OO DBMS would establish the subdatabase by identifying all the extensional patterns that belong to the specified types and then perform the operation(s). The implementation of OQL on a SUN workstation is reported in a master's thesis [TY88].

A query block in OQL consists of a Context clause and an Operation clause. The Context clause has two optional subclauses: a Where subclause and a Select subclause. This structure is shown below.

```
context association pattern expression
   where conditions
   select object classes and/or attributes
operation(s) object class(es)
```

In the context clause, the user specifies a desired subdatabase by specifying its intensional pattern and extensional pattern types of interest (both are specified in the association pattern expression). A linear association pattern expression has the form "Class1 [intra-class conditions] **op** Class2 [intra-class conditions] **op** Class3 [intra-class conditions] ..." where "op" is one of the association pattern operators. Each operator separates two E-classes that are directly associated in a schema. The intra-class conditions enclosed in brackets following a class name are optional and are expressed in the form of predicates that involve the descriptive attributes of that class.

1. We note that Section s3 is related to more than one Course instance and Section s4 is not related to any Course instance. Naturally, there is a constraint on the database that restricts the mapping between Section and Course to N:1 and another Non-null constraint on the aggregation association of Course with Section. We assume that these constraints are waived here in order to describe the most general case.

The Where subclause further causes the extensional patterns that do not satisfy some conditions to be dropped from the Context subdatabase (the subdatabase defined by the Context expression). The conditions that can be specified in the Where subclause are inter-class comparison conditions, which are comparisons between some descriptive attributes of two classes if these attributes are type comparable, and/or comparison conditions that involve aggregation functions (e.g., COUNT).

The Select subclause identifies the descriptive attributes and/or classes in the Context subdatabase that are to be operated on by the operation(s) specified in an operation clause. It eliminates attributes and classes that are not relevant to the operations. An operation, in the Operation clause can be either a system-defined data manipulation operation (e.g., Display, Update, or Print) or a user-defined operation (e.g., Rotate, Order-part, or Hire-employee). If the Display (Print) operation is specified in the operation clause, it causes the values of the descriptive attributes identified by the Select subclause to be displayed (printed) in a tabular form.

The operators that can be used in the association pattern expression of the Context clause are the **association operator** and the **non-association operator**. In the following we give a brief description of the association operator only, which will be used by the rule-based language to be introduced in Section 4. In the remainder of this paper, capital letters are used to denote E-classes (A, B, ...) and small letters with an integer appended to each letter are used to denote objects (e.g., a1, a2, ... and b1, b2, ... are objects that belong to the classes A and B, respectively).

When the association operator (*) is applied to two directly associated E-classes A and B in a database (i.e., the expression "A * B"), it returns a subdatabase whose intensional pattern consists of the two classes A and B and their association. The resulting subdatabase contains also the set of extensional patterns drawn from the operand database such that each extensional pattern contains objects of both A and B (i.e, extensional patterns that are of the type <A,B>). B objects that are not associated with any A object and A objects that are not associated with any B object in the operand database are not retained in the result. The following example queries illustrate the use of the association operator.

Query 3.1 Display the names of the teachers who teach some sections and the section#'s of these sections.

```
context Teacher * Section
   select name, section#
display
```

If the Context expression of this query is applied to the subdatabase SDB of Figure 3.1, it will return the subdatabase shown in Figure 3.2. The intensional pattern of this subdatabase consists of the classes Teacher and Section and their association (the descriptive attributes of the classes Teacher and Section are also represented in the figure). The set of extensional patterns of the resulting subdatabase is {<t1,s2>, <t2,s3>, <t3,s4>}. The extensional pattern <t4> (or <t4, Null>) is not included in the result because its Section-component is Null (similarly, the pattern <s5> is not included). The result of the Display operation is a binary table in which each tuple contains a name value and a section# value.

The definition of the association operator can be easily generalized to the case when the association pattern expression contains more than two classes. For example, the expression "A * B * C" returns the extensional patterns that are of the type <A,B,C>. It is noted here that one can define a single extensional pattern type using the association operator. A mechanism for defining a richer variety of extensional pattern types in a single expression is described in Section 5.

Query 3.2 Print the Department names for all departments that offer 6000 level courses that have current offerings (sections). Also, print the titles of these courses and the textbooks used in each section.

```
context Department * Course [6000 <= c# < 7000] *
        Section
    select name, title, textbook
print
```

In this query, the intra-class condition on the C# attribute of Course is enclosed in brackets following the class name in the Context expression.

We note here that the association operator can be used between any two classes whether they are connected by a generalization or an aggregation association. The types of associations between classes are explicitly defined in the schema and restating these association types in queries is unnecessary. The query processor of an OO DBMS can make use of the type information stored in the dictionary to properly interpret the queries and enforce the relevant semantics and constraints. For example, a link that exists between an instance of the class TA and an instance of the class Grad of Figure 2.1 is an identity link. In other words, the semantics implied by the generalization association here is that the two instances are actually two different perspectives of the same real world object.

OQL makes full use of the inheritance property of the generalization association. Therefore, the expression "RA * Section" is a legal expression since the class RA inherits the aggregation association with Section (Figure 2.2) along a unique generalization path. However, in some cases, a class inherits the status of being related to another class along different generalization paths. For example, the class TA inherits the status of being related to Section from both Teacher and Grad with each of them having its distinctive meaning. In this case, at least one of the classes along the intended generalization path has to be explicitly referenced in the association pattern expression to resolve the ambiguity. Thus, the ambiguity in the expression "TA * Section" is resolved by using either one of the two expressions "TA * Grad * Section" or "TA * Teacher * Section."

4. The OO Deductive Rule-based Language

Our rule-based language can be used to derive new subdatabases from other existing or derived subdatabases and therefore the world of subdatabases is closed under this language. A derived subdatabase is called the **target subdatabase** and the subdatabases used to derive it are called the **source subdatabases.** A target subdatabase that is derived by a rule can be a source subdatabase of another rule. We refer to a class in a target subdatabase as a **target class**. The class in the source subdatabase from which a target class is derived is called the **source class.** The set of instances of a target class is a subset of the set of instances of the source class from which it is derived.

Before introducing our deductive rule-based language, we first extend the definition of a subdatabase as given in [ALA89a] (see Section 3) by introducing the induced generalization association construct. Our objective is to provide a conceptual description for the relationships (associations) that may exist between classes even if they belong to different subdatabases. This enables using the association operator between any two associated classes that belong to two different subdatabases and therefore a rule can derive a new subdatabase out of more than one source subdatabases.

4.1 The Induced Generalization Association

Between every target class and its source class, there is a generalization association that is **induced** by the deductive rule, which emanates from the source class and connects to the target class. Therefore, a target class inherits all the aggregation associations of its source class, which establishes the inter-subdatabase connections as shown in the following example.

Let SD be a subdatabase whose intensional pattern is as shown in Figure 4.1 (SD could have been derived from the original database by a deductive rule). Let SD1 and SD2 be two subdatabases (Figure 4.1) that are derived from SD by two rules. Class A that appears in SD1, which we refer to as SD1:A (i.e., by qualifying the class name with the subdatabase name using a colon), is derived from the corresponding class in SD, i.e., SD:A. Therefore, there is an induced generalization association that emanates from the source class SD:A and connects to the target class SD1:A, as shown in the figure. Similarly, there is an induced generalization association emanating from SD:D and connecting to SD2:D and two induced generalization associations emanating from SD:C and connecting to the classes SD1:C and SD2:C.

Each of the classes of SD1 and SD2 inherits all the aggregation associations of its superclass in SD. For example, SD1:A inherits the links that connect to and the links that emanate from its superclass SD:A, which effectively means that there are aggregation associations between SD1:A and each of the classes SD:B and SD:C. The inherited association that connects SD1:A to SD:C is further inherited by SD1:C, which is a subclass of SD:C. Since this association that connects SD1:A to SD1:C is local (internal) to the subdatabase SD1, it is explicitly represented in its intensional pattern (thus an intensional association pattern of a subdatabase consists of the classes that appear in the subdatabase and their local associations). Similarly, the inherited association that emanates from SD1:A and connects to SD:C is further inherited by SD2:C. This means that there is an (inherited) association between the two classes SD1:A and SD2:C, which belong to two different subdatabases. Thus, the inheritance property of the induced generalization association establishes the connections between classes from different subdatabases. Figure 4.2 shows an equivalent representation to that of Figure 4.1, where all the aggregation associations that the class SD1:A inherits from its superclass SD:A are explicitly represented.

By considering the definition of the induced generalization association construct as introduced above, each of the classes of the subdatabase SDB of Figure 3.1 inherits all the aggregation associations of its source classes in the original database of Figure 2.1. For example, the class SDB:Course inherits the aggregation associations to the E-classes Department, Transcript, and Course (through the Prereq. association) and to the D-classes C#, Title, and Credit-hours from its super class Course in the original database. In the remainder of this paper, if a class name is referenced in any expression without qualifying it by a subdatabase name, the "base" class (i.e., the class that appears in the original database) is assumed.

The functionality of the association operator (*) can now be extended in such a way that it can be used to operate on two classes even if they belong to two different subdatabases provided that there is an (inherited) association between them. For example, the expression "SD1:A * SD2:C" is a legal association pattern expression against the classes of Figure 4.2, which creates a new subdatabase, say X, that consists of the classes A and C. The classes X:A and X:C are subclasses of the classes SD1:A and SD2:C, respectively. Only the instances of the classes A and C that are associated and that appear in the subdatabases SD1 and SD2, respectively, will appear in the subdatabase X.

4.2 Syntax and Semantics of Deductive Rules

A rule in our OO rule-based language has an If-Then structure as follows.

if context association pattern expression
  where conditions
then subdatabase-id (classes)

The Context clause and its optional Where subclause are the same as described in Section 3 above. The subdatabase-id in the Then clause is a unique name to be given to the derived subdatabase. The intensional pattern of the derived subdatabase consists of the classes referenced in the argument list following the subdatabase-id in the Then clause. These classes should be a subset of the classes referenced in the association pattern expression of the If clause. Other unreferenced classes will not be retained in the derived subdatabase. The extensional patterns, on the other hand, are derived from the extensional patterns that satisfy the conditions of the If clause and its Where subclause. In other words, the Then clause derives new patterns of object associations among objects of some specified classes if these objects exist in some "base" or other derived association patterns. The following rule provides a simple example.

R1 Derive the subdatabase Teacher-course in which only the classes Teacher and Course appear such that a Teacher instance teaches a Course instance (i.e., teaches one of the sections of the course).

If context Teacher * Section * Course
then Teacher-course (Teacher, Course)

If this rule is applied to the Subdatabase SDB of Figure 3.1, it returns the subdatabase Teacher-course whose intensional pattern and set of extensional patterns are shown in Figure 4.3. The intensional pattern of Teacher-course consists of the classes Teacher and Course only. The class Section is not retained in this subdatabase because it is not referenced in the argument list following the subdatabase name in the Then clause. Since Teacher and Course in the operand database are not directly associated but are associated through Section, a new direct association is derived between them in the resulting subdatabase (Figure 4.3a).

At the extensional level new direct links are inferred between the instances of Teacher and Course. In other words, the derived subdatabase contains a set of extensional patterns such that an extensional pattern consists of a Teacher instance and a Course instance with a direct link between them, provided that the two instances satisfy the associativity condition that is specified in the If clause, i.e., the two instances must coexist in the operand database in an extensional pattern that also contains a Section instance. For example, a direct association is derived between the instances t1 and c1 as shown in Figure 4.3b because t1 and c1 are associated through s2 in the operand subdatabase SDB of Figure 3.1.

If a target class in a derived subdatabase is to inherit only a subset of the descriptive attributes of its source class, then these attributes should be listed in brackets following the class name in the Then clause, otherwise all attributes are inherited (i.e., the default is "all attributes"). For example, if the class Teacher in the subdatabase Teacher-course is to inherit only the attributes SS# and Degree, the above rule will be expressed as

If context Teacher * Section * Course
then Teacher-course (Teacher [SS#, Degree], Course)

In this case, the attribute Name will not be accessible from the class Teacher-course:Teacher.

The following are some additional rules defined for the University database of Figure 2.1. Each of these rules derives a new subdatabase and illustrates certain aspects of the semantics of our rule-based language.

R2 If the total number of students who are enrolled in a course that belongs to the CIS department is greater than 39, then suggest offering the course in the next semester.

If context Department [name = 'CIS'] * Course *
    Section * Student
  where COUNT (Student **by** Course) > 39)
then Suggest-offer (Course)

COUNT is an aggregation function that returns the number of students associated with each course (i.e., via some sections). The If clause identifies the set of extensional patterns that satisfy the association pattern expression and the condition stated in the Where subclause. In other words, this set of extensional patterns does not include the extensional patterns that contain courses with less than forty enrolled students. The derived subdatabase Suggest-offer contains only the class Course together with the Course instances that

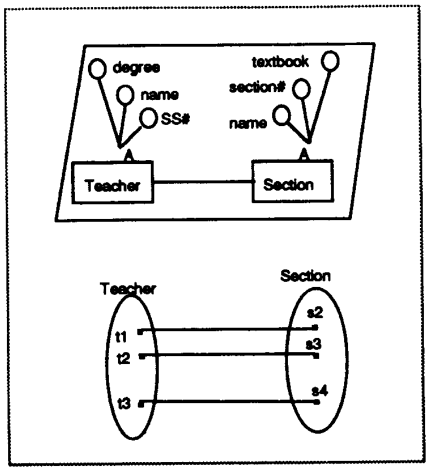

Figure 3.2: The subdatabase defined by a Context expression

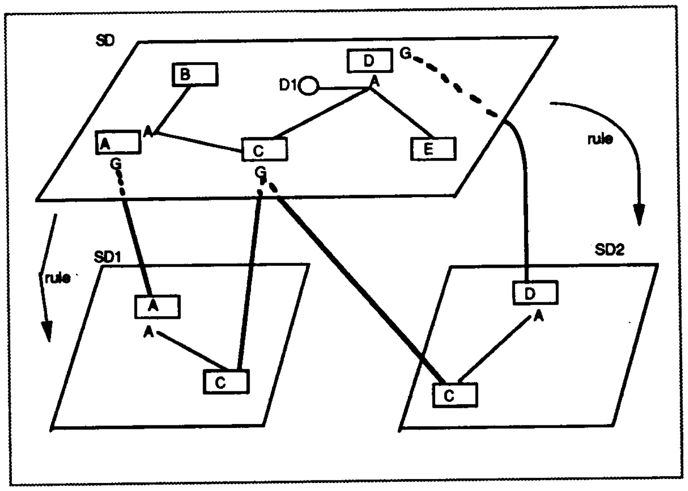

Figure 4.1 Subdatabases SD1 and SD2 derived from SD

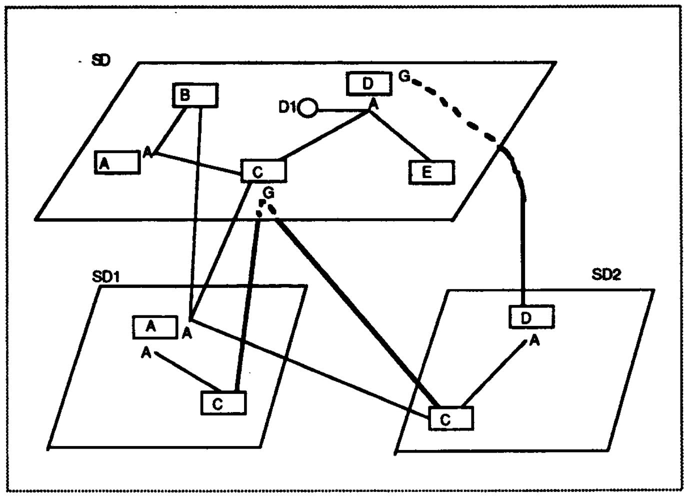

Figure 4.2 The Associations inherited by SD1:A from SD:A
are explicitly represented

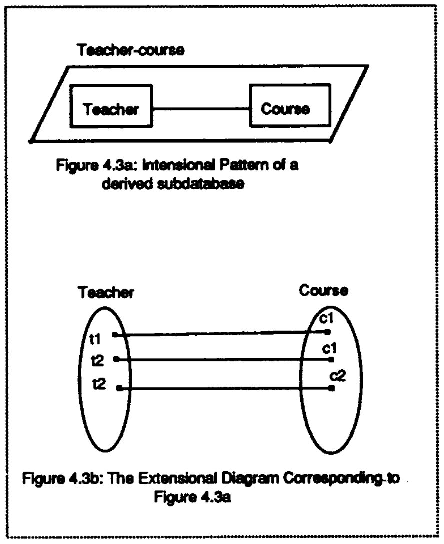

Figure 4.3a: Intensional Pattern of a derived subdatabase

Figure 4.3b: The Extensional Diagram Corresponding to Figure 4.3a

Figure 4.3: The Intensional Pattern and Extensional Diagram for a Subdatabase that is derived by a rule

satisfy the conditions stated in the If clause (i.e., the instances that appear in the extensional patterns identified by the If clause). Because the closure property is preserved, this subdatabase can be operated on by other deductive rules in the same manner to further derive new subdatabases as shown by the following rule.

R3 If, for any department, the number of courses that are suggested to be offered next semester is greater than 20 courses, then the department needs more resources (i.e., budget, rooms, labs, etc.).

If context Department * Suggest-offer:Course
  where COUNT (Suggest-offer:Course **by** Department) > 20
then Deps-need-res (department)

COUNT here returns for each department the number of courses that belong to it and that are suggested to be offered as determined by the previous rule. Since the class Course of the subdatabase Suggest-offer is a subclass of the base class Course (i.e., there is an induced generalization association between them), it inherits the aggregation link to the base class Department, hence, the expression "Department * Suggest-offer:Course" is a legal expression. This rule derives a new subdatabase called Deps-need-res (departments that need resources). The source subdatabases from which this subdatabases is derived are: the original database and the Suggest-offer subdatabase.

R4 If a graduate student is currently teaching a course that is suggested to be offered, then he/she may teach the same course in the next semester.

If context TA * Teacher * Section * Suggest-offer:Course
then may-teach (TA, Course)

R5 A graduate student may teach an undergraduate course (i.e., c# < 5000) if he/she has taken the course and got a grade of B or more.

if context Grad * Transcript [grade => 'B'] *
  Course [c# < 5000]
then may-teach (Grad, Course)

Rules R4 and R5 derive extensional patterns into the same subdatabase (May-teach) but based on different conditions. Thus, if both rules are applied, May-teach will contain the union of the two sets of extensional patterns derived by the two rules.

4.3 Querying the Derived Data

Once the deductive rules that derive new subdatabases are defined, the classes of the derived subdatabases can be referenced in association pattern expressions in any OQL query in the normal way. For example, the following is an OQL query that operates on the base classes Faculty and Advising and on the class TA of the subdatabase May-teach.

Query 4.1 For the teaching assistants who may teach a course in the next semester, have advisors, and whose GPA's are less than 3.5, display their names and their advisors' names.

context Faculty * Advising * May-Teach:TA [GPA < 3.5]
  select TA [name], Faculty [name]
display

The class May-teach:TA is a subclass of the base class TA, which in turn is a subclass of the class Grad. Thus, May-teach:TA inherits the aggregation association with Advising from Grad and consequently the expression "Advising * May-teach:TA" in the above query is legal. As is the case in the above query, an attribute that appears in the Select subclause has to be qualified by its class name if it is not unique among the attributes of the classes that are referenced in the Context clause. This is done by enclosing the attribute name in brackets following the class name (e.g., "TA [name]" in the above query).

In the backward chaining strategy (backward and forward chaining strategies are described in Section 6), this query is evaluated as follows. Since TA is referenced in the query in the context of May-teach, rules R4 and R5 will be triggered for execution to derive the subdatabase May-teach. But, in order to derive May-teach, the subdatabase Suggest-offer (which is referenced by rule R4) must be derived. This causes rule R2 that derives Suggest-offer to be triggered for execution. R2 does not refer to any other derived subdatabase, hence, the expressions of R2 are evaluated against the base classes. The result is then fed to rule R4 that participates in deriving May-teach. (similarly, the expressions of R5 are evaluated against the base classes.) The subdatabase May-teach is then used to evaluate the given query.

5. Transitive Closure

The transitive closure operation is expected to be a fundamental operation in future database systems and knowledge systems. Before describing how it can be performed in our language we describe a mechanism for defining several extensional pattern types in a single association pattern expression. This mechanism is then used in deductive rules that perform the transitive closure operation.

5.1 Association Pattern Subexpressions

We allow more than one extensional pattern type to be identified by a single association pattern expression by enclosing a subexpression of it inside braces. This subexpression identifies a certain extensional pattern type. For example, the expression "A * {B * C} * D" returns the subdatabase whose intensional pattern consists of the referenced four classes and whose set of extensional patterns includes all patterns that are of the types <A,B,C,D> and <B,C>. In other words, this expression means to select both the instances of A,B,C and D classes that are connected (associated) all the way through as well as those instances of B and C that are connected to each other but not necessarily connected to the instances of A and/or B. The braces around B * C capture the semantics of the Outerjoin concept introduced in [COD79].

If, in the above expression, an extensional pattern of the type <B,C> appears in the resulting subdatabase as part of a larger pattern of the type <A,B,C,D>, it will not appear independently in that resulting subdatabase. For example, if the original database contains only the two patterns

<a1,b5,c5,d5> and <a3,b2,c2>, then the expression "A * {B * C} * D" returns the extensional patterns <a1,b5,c5,d5> and <b2,c2>. The extensional pattern <b5,c5> will not appear independently in the result since it already appears as a part of the extensional pattern <a1,b5,c5,d5>. In general, an extensional pattern of a certain specified type will not appear independently in the result if it is part of a larger extensional pattern.

Subexpressions can be nested to several levels. For example, the expression "{{{A} * B} * C} * D" identifies the extensional pattern types <A>, <A,B>, <A,B,C>, and <A,B,C,D>.

<u>Query 5.1</u> Display the SS#'s of <u>all</u> graduate students (whether they have advisors or not) and for <u>those</u> graduate students who have advisors, display their advisors' names.

context { Grad } * Advising * Faculty
  select Grad [SS#], Faculty [name]
display

The result of the display operation is a binary table each tuple of which contains a Grad's SS# and either a faculty name or a Null value if the student has no advisor.

5.2 Transitive Closure Operation

Different aliases (range or iteration variables) of a class can be automatically generated in OQL by appending an Underscore and an integer to the class name in the association pattern expression (e.g., Grad_1 is a an alias of Grad). This feature is used in the definition of the transitive closure operation that follows.

The transitive closure operation is performed in our language by iterating over some classes and associations that form a cycle. Let A, B and C be three classes in a schema that form a cycle, i.e., A has associations with both B and C and B has an association with C. The following rule derives a subdatabase X that contains pairs of instances of the class A that are associated with each other via some B and C instances. In other words, each extensional pattern in this subdatabase contains two objects from the class A.

if context A * B * C * A_1
then X (A, A_1)

By extending the Context expression in this rule to form a second iteration, i.e., the expression "{A * B * C * A_1} * B_1 * C_1 * A_2", one can derive a subdatabase that contains a three-level hierarchy of A instances. The braces are used in this expression to keep the first and second level A instances that are associated with each other even if they are not associated with third level A instances.

This iteration can be expressed in our language by adding the "@" sign as a superscript at the end of an association pattern expression that forms a cycle. An optional number N (Number of iterations) following the @-sign causes the underlying system to traverse the cycle N times. (For a certain hierarchy of instances, iteration stops when Null values are encountered or at the Nth iteration, i.e., Nth descendant from the root of the hierarchy.) If such a number is not present, the cycle is traversed until Null values are obtained for all the hierarchies of instances (i.e., the transitive closure operation is performed). Using this technique, the two expressions "{A * B * C * A_1} * B_1 * C_1 * A_2" and "{{A * B * C * A_1} * B_1 * C_1 * A_2} * B_3 * C_3 * A_3" are represented as (A * B * C)$^{@2}$ and (A * B * C)$^{@3}$, respectively. Our approach as described above allows for representing the transitive closure operation in the form of looping rather than in a recursive form which is a simpler representation especially for end-users. The following are two example deductive rules.

<u>R6</u> a graduate student may be taught by other graduate students in some sections and also may teach other graduate students in some other sections. Derive the Grad-teaching-grad hierarchy. (It is assumed here that the relationship between the instances of the class Grad is not cyclic, i.e., if Grad g1 teaches Grad g2 or teaches any of the teachers of g2, then g2 cannot teach g1 nor any of the teachers of g1).

if context (Grad * TA * Teacher * Section * Student)$^{@}$
then Grad-teaching-grad (Grad, Grad_@)

The intensional pattern of the resulting subdatabase Grad-teaching-grad consists of the classes Grad, Grad_1, Grad_2, ... until Null values in all the hierarchies are encountered (the second argument to Grad-teaching-grad, i.e., Grad_@, stands for Grad_1, Grad_2, ...). In other words, the intensional pattern of the derived subdatabase is determined at run time.

<u>R7</u> Derive a subdatabase which contains only the 1st level and 3rd level in the grad-teaching-grad hierarchy.

if context (Grad * TA * Teacher * Section * Student)$^{@2}$
then first-and-third (Grad, Grad_2)

6. Control Strategies

The two control strategies used in inferencing are forward and backward chaining of rules. In the backward chaining strategy, the evaluation of a derived subdatabase is delayed until a retrieval query that needs the derived data is issued. On the contrary, in the forward chaining strategy, an up-to-date copy of the derived subdatabase is always kept available, which improves the performance of retrieval operations. Whenever the data that is used to derive a subdatabase is updated (e.g., by associating, dissociating, inserting objects, etc.), the relevant deductive rules are run to maintain the consistency between the derived subdatabase and the original database.

In the relational system POSTGRES [STO87], only one of these two control strategies (i.e., forward and backward chaining of rules) is assigned to each rule in the system. A rule that is defined to follow the forward chaining strategy (i.e., a forward chaining rule) will be executed whenever the data that is read by the rule is updated, also, an up-to-date copy of the derived data is explicitly stored. A rule that is defined to follow the backward chaining strategy (i.e., a backward chaining rule) will be triggered for execution whenever the data that the rule derives is requested (i.e., in a query) but the derived data is not preserved after the query session. In this rule-oriented control strategy, a rule is restricted to follow only one of the two control strategies at all times.

The disadvantage of this rule-oriented control strategy is that it imposes a restriction on the mixing of forward and backward chaining rules such that a forward chaining rule cannot read any data written by backward chaining rules [STO87]. To describe this problem, let the following be a series of rules Ra to Rd and the results REa to REd derived by these rules.

$$DB \xrightarrow{Ra} REa \xrightarrow{Rb} REb \xrightarrow{Rc} REc \xrightarrow{Rd} REd$$

Also, let Ra and Rb be defined as backward chaining rules and Rc and Rd as forward chaining rules. If the original database DB is updated, rules Rc and Rd, though they are forward chaining rules, will not be triggered to update the result REd until someone requests the data of REb. Thus, REd may be inconsistent with the base data.

To overcome this problem, we use a result-oriented control strategy in which we specify for each result (derived subdatabase) whether it is to be **pre-evaluated** or **post-evaluated**. The same rule may follow the forward or backward chaining strategy depending on whether the derived subdatabase is to be pre- or post-evaluated.

To illustrate by the example above, assume that REd is defined as pre-evaluated and REb is defined as post-evaluated. Whenever the database DB is updated, the rules Ra, Rb, Rc, and Rd will be triggered in the forward chaining fashion to keep REd (which is explicitly stored) up-to-date. REb, on the other hand, will be evaluated whenever a retrieval operation is issued against it. In this case, the rules Ra and Rb that derive REb are applied in the backward chaining fashion. Thus, Ra and Rb follow one control strategy when deriving REd and the other control strategy when deriving REb. This technique offers more flexibility and alleviates the restriction in POSTGRES described above.

7. Conclusion

In this paper, we have introduced the induced generalization association construct and presented a deductive rule-based language for object-oriented databases. The world of subdatabases is closed under this language, which facilitates defining inference chains in which each rule derives a new subdatabase based on the subdatabases derived by previous rules in the chain. The transitive closure operation can be specified in our language in the form of looping rather than in a recursive form. A result-oriented control strategy to be used as the underlying implementation technique has also been introduced in this paper.

ACKNOWLEDGEMENTS

Research on the rule-based language was supported by the U.S. West Advanced Technologies grant number UPN 88071315. Work on the Object-Oriented Query Language OQL was supported by the Navy Manufacturing Technology Program through the National Institute of Standards and Technology (formerly the National Bureau of Standards) grant number 60NANB4D0017 and by the National Science Foundation grant number DMC-8814989. The development efforts are supported by the Florida High Technology and Industry Council grant number UPN 85100316.

BIBLIOGRAPHY

ALA89a A.M. Alashqur, S.Y.W. Su, and H. Lam, "OQL: A Query Language for Manipulating Object-oriented Databases," Accepted for Publication, the 15th VLDB Int. Con., 1989.

ALA89b A.M. Alashqur, A Query Model and Query and Knowledge Definition Languages for Object-Oriented Databases, a Ph.D. Thesis, University of Florida, 1989.

BAN87 Jay Banerjee, et al., "Data Model Issues for Object-Oriented Aplications," ACM Trans. on Office Information Systems, January 1987.

BAT85 D. Batory and W. Kim, "Modeling Concepts for VLSI CAD objects," ACM TODS, September 1985, pages 322-346.

CER86 Stefano Ceri, George Gottlob, and Gio Wiederhold, "Interfacing Relational Databases and Prolog Efficiently," Proc. of the 1st Intl. Con. on Expert Database Systems, 1986.

CHA84 C. L. Chang and A. Walker, "PROSQL: a PROLOG Programming Interface with SQL/DS," Proceedings of the 1st Intl. Workshop on Expert Database Systems, 1984.

COD79 E. Codd, "Extending the Database Relational Model to Capture More Meaning," ACM TODS, Vol. 4, No. 4, 1979.

DEL88 Lois M.L. Delcambre and James N. Etheredge, "A self-Controlling Interpreter for the relational Production Language," Proceedings of ACM SIGMOD Conference on Management of Data 1988, pages 396-403.

DIT86 K.R. Dittrich, "Object-oriented Database Systems: the Notion and Issues," Proc. of the Intl. Workshop on Object-Oriented Database Systems, California, September 1986.

FIS87 D.H. Fishman, et al., "Iris: An Object-Oriented Database Management System," ACM Transaction on Office Information Systems, January 1987, Pages 48-69.

FOR88 S. Ford, et al., "Zeitgeist: Database support for object-oriented rogramming," in the Proceedings of the Second International Workshop on Object-Oriented Database Systems, 1988.

GAL84 Herve Gallaire, Jack Minker, and Jean-Marie Nicolas, "Logic and Databases: A Deductive Approach," ACM Computing Surveys, June 1984, Pages 153-185.

HAM81 M. Hammer and D. McLeod, "Database Description with SDM: A Semantic Association Model," ACM TODS, September 1981.

HUL87 R. Hull and R. King, "Semantic Database Modeling: Survey, Applications, and Research Issues," ACM Computing Surveys, September 1987.

JAR84 Matthias Jark, Jim Clifford, and Yannis Vassiliou, "An Optimizing Prolog Front-End to a Relational Query System," Proc. of ACM SIGMOD Con. on Management of Data 1984.

LAM89 H.M. Lam, S. Su, and A.M. Alashqur, "Integrating the Concepts and Techniques of Semantic Data Modeling and the Object-oriented Paradigm," Proc. of the 13th Intl. Computer Software and Applications Conference (COMSAC 89), 1989.

MAI88 Christophe de Maindreville and Eric Simon, "A Production Rule Based Approach to Deductive Databases," Proc. of the 4th Intl. Con. on Data Engineering, California, 1988.

RAS88 L. Raschid and S.Y.W. Su, "A Transaction-Oriented Mechanism to Control Processing in a Knowledge Base Management System," Proc. of the Intl. Con. on Expert Database Systems, 1988.

STO87 Michael Stonebraker, Eric Hanson and Chin-Heng Hong," The Design of the POSTGRES Rules System," Proc. of the 3rd Intl. Con. on Data Engineering, California, 1987.

SU89 S.Y.W Su, V. Krishnamurthy, and H. Lam, "An Object-oriented Semantic Association Model (OSAM*)," appearing in: A.I. in Industrial Engineering and Manufacturing: Theoretical Issues and Applications, S. Kumara, et al. (eds.), American Institute of Industrial Engineering, 1989.

TY88 Frederick Ty, "G-OQL: Graphics Interface to the Object-Oriented Query Language OQL," Master thesis, University of Florida, 1988.

ULL85 Jeffrey Ullman, "Implementation of Logical Query Languages for Databases," ACM TODS, September 1985.

VAS84 Y. Vassiliou, J. Clifford, and M. Jark, "Access to Specific Declarative Knowledge by Expert Systems: The Impact of Logic Programming," Decision Support Systems 1, 1, 1984.