An Association Algebra For Processing Object-Oriented Databases †

Mingsen Guo * Stanley Y.W. Su ** Herman Lam **

* Bull HN Worldwide Information Systems, Phoenix, AZ 85000
** Database Systems R&D Center, University of Florida, Gainesville, FL 32611

Abstract: This paper presents an association algebra (A-algebra) for manipulating O-O databases which is analogous to the relational algebra for relational databases. In this algebra, objects and their associations in an O-O database are uniformly represented by association patterns and are manipulated by a number of operators. These operators are defined to operate on association patterns of both heterogeneous and homogeneous structures. Very complex structures (e.g., network structures of object associations across several classes) can be directly manipulated by these operators. Therefore, the association algebra has greater expressive power than the relational algebra which manipulates on relations of compatible structures. Some mathematical properties of these operators are described in this paper together with their application in query decomposition and optimization. The algebra has been used as the basis for the design and implementation of an O-O query language called OQL and a knowledge rule specification language.

1. Related Work and Motivations

Object-Oriented (O-O) and semantic data models [HAM81, BAT84, KIN84, DAD86, MAI86, MAN86, SU86, ZDO86, BAN87, FIS87, KIM87, ROW87, ZDO87, CAR88, CHR88, COL89, SU89] offer more powerful constructs for modeling the structural and behavioral properties of objects found in advanced applications such as CAD/CAM, CASE, and decision support. For these models to be truly useful, they must provide some object manipulation languages, which can take advantage of the expressive power of the models and provide the users with simple and powerful querying facilities. Recently, several query languages such as DAPLAX [SHI81], GEM [ZAN83], ARIEL [MAC85], FAD [BAN87], POSTQUEL [ROW87], EXCESS [CAR88], and others reported in [DAD86, MAN86, BAN87, FIS87, BAN88, COL89, SHA90] have been proposed. These languages were developed based on different paradigms. For example, DAPLAX and the query language of [MAN86] are based on the functional paradigm. The query language of [BAN88] is based on the message passing paradigm. Other query languages are based on the relational paradigm: an extension of QUEL [ROW87, CAR88]; an extension of SQL [DAD86]; and an extension of the relational algebra [COL89]. The query language of [FIS87] is based on both functional and relational paradigms, allowing functions to be used in Object-oriented SQL (OSQL) constructs.

The above languages have O-O flavor and have taken significant steps towards the development of a powerful O-O query language. Query languages such as DAPLAX [SHI81], GEM [ZAN83], ARIEL [MAC85], and the object-oriented query language described in [BAN88] are based on an object-oriented view of a database defined in terms of objects, object classes, and their associations. A query in these languages is formulated by specifying a class (usually a nonprimitive-class, whose instances are real world objects) in the schema as a central class with some path expressions. Each path expression starts from the central class and ends at another class (usually a primitive-class, whose instances are of basic data types such as integer, string, set, and etc.). A restriction condition can be specified on the class referenced at the end of a path expression. This class can also be specified in the list of attributes to be retrieved. The result of a query is a set of tuples, each of which corresponds to a single instance of the central class and contains values related to that instance which are collected from classes specified in the list.

A major drawback of these query languages is that they do not maintain the closure property [ALA89]. In these languages, the input to a query has an O-O representation (i.e., a network of objects, classes, and their associations) whereas its output is a relation which does not have the same structural and behavioral properties as the original objects. Consequently, the result of a query cannot be further processed by the same set of language operators.

The query languages proposed in [DAD86, MAN86, BAN87, ROW87, CAR88, COL89] use nested relations as their logical views of O-O databases. Although these languages are closed, i.e., operators in these languages operate on nested relations and produce nested relations, the nested relations, in our view, is not a proper logical representation for an O-O database which is basically a network structure of object associations. Mapping from a network representation to nested relations is an extra process. Furthermore, the relationships among objects in O-O databases are not restricted to plane graphs. It can be as complicated as a surface graph. In order to use a nested relation to represent these complex structures, a large amount of data has to be replicated in the representation.

A query algebra [SHA90] was proposed recently based on the O-O model ENCORE [ELM89]. Although ENCORE models applications as networks of objects, object types, and their associations, the domain of the algebra is defined as sets of objects of the Tuple type, which is essentially the nested relation representation since it allows the nesting of tuples. Therefore, the mapping problem addressed above still remains. In this algebra, two identical queries or two identical operations in a single query do not give the same response, since each result collection is a newly identified object in the database. To eliminate duplicated copies of the same newly created object, the algebra introduces operations named DupEliminate and Coalesce, which would not be necessary if the algebra were to directly support the network structured processing of O-O databases. We further observe that the union operation in this algebra may produce a collection of objects having the

† This research is supported by the National Science Foundation (DMC-8814989) and the National Institute of Standard and Technology (60NANB4D0017). The development effort is supported by the Florida High Technology and Industrial Council (UPN88092237).

same data type but with different structures (e.g., the union of two collections of objects of the Tuple type with different arities). Nevertheless, the other operators introduced in this algebra are not defined to operate on collections of objects with heterogeneous structures.

A common limitation of many existing query languages is that they cannot express "non-association" relationship between objects easily, i.e., identify objects in two classes that are not associated with each other while their classes are. For example, in an O-O database, Suppliers s1 and s2 supply Parts p1 and p2, respectively. GEM, POSTQUEL, and several other query languages provide the "dot" construct (Suppliers.Parts) and ARIEL provides the "of" construct (Parts of Suppliers) to navigate from class Suppliers to class Parts to produce object pairs (s1,p1) and (s2,p2). However, they do not have a language construct for specifying the semantics that s1 does not supply p2 and s2 does not supply p1. Similarly, in functional languages, only the function Parts(Suppliers) is provided to specify the associations of (s1,p1) and (s2,p2) but not the non-association of these suppliers and parts.

To conclude our survey of the related works, we would like to stress the importance of using graphs as the logical representation of an O-O database at both intensional and extensional levels as exemplified by O_2 [LEC88], FAD [BAN87], and OSAM* [SU89]. The query language and its underlying algebra should provide constructs to directly process graphs with different degree of complexity. They should also support the specification of non-associations and the processing of heterogeneous structures. Furthermore, the closure property should be maintained.

In this paper, we propose an Association Algebra (A-algebra), which uses graph as the representation of O-O databases and operates on the graph. Analogous to the development of the relational algebra for relational databases, the development of the A-algebra provides the formal foundation for query processing and optimization in O-O databases and for designing O-O query languages. Unlike the record(tuple)-based relational algebra [COD70 and COD72] and the query algebra [SHA90], the A-algebra is association-based, i.e., the domain of the algebra is sets of association patterns (e.g., linear structures, trees, lattices, networks, etc.) and processing an O-O database is based on the matching and manipulation of heterogeneous or homogeneous patterns of object associations. Operators of the A-algebra can be used to navigate along the path of interest to construct a pattern with a complicated structure from those with simple structures, or to decompose a complicated one. Nine operators have been defined: two unary operators [A-Select (σ) and A-Project (Π)], and seven binary operators [Associate (*), A-Complement (|), A-Union (+), A-Difference (–), A-Divide (÷), NonAssociate (!), and A-Intersect (•)], where the prefix A- stands for "Association". Although, many of these operators correspond to the relational algebra operators, they are different from them in that they operate on complicated heterogeneous structures. In this respect, the A-algebra is more general than the relational algebra.

Based on the A-algebra, an O-O Query Language (OQL) [ALA89a, ALA89b] and a knowledge rule specification language [ALA90] for the O-O semantic data model OSAM* [SU89] have been developed and implemented [SU88, LAM89, PAN89, CHU90, SIN90].

The rest of this paper is organized as follows. Some basic concepts in association-based query formulation are described in Section 2 with the help of examples. In Section 3, after giving the formal definitions of Schema Graph (**SG**), Object Graph (**OG**), and association pattern, the formal definitions of the association operators are presented. Simple mathematical properties are given after the definition of each operator. The A-algebra expressions for some example queries are given to demonstrate the utility of the algebra. In Section 4, additional properties of the association operators are presented. Finally, the conclusion is given in Section 5.

2. Association-based Query Formulation

O-O semantic data models provide a conceptual basis for defining O-O databases. Although each model has some unique constructs, there are several common structural and behavioral properties based on which an algebra can be developed and used to support these models. First, an object is assigned a system-wide unique object identifier (**OID**) so that it can be unambiguously distinguished and referenced. Second, objects having the same structural and behavioral properties are grouped together to form an object class. Third (and very important), object classes and their objects are inter-related (i.e., associated) with other classes and their objects in the database. Different O-O models recognize different types of associations. For example, two of the most commonly recognized associations are **aggregation** and **generalization**. Aggregation captures the semantics of **is–part–of**, **is–composed–of**, or **has–properties–of**. It allows an object class (and its objects) to be defined in terms of other object classes (and their objects). Generalization captures the semantics of **is–a** or **is–a–kind–of** relationship. Objects in a subclass inherit the structural and behavioral properties of its superclasses.

The semantics of these and other association types can be expressed in terms of constraints governing the data manipulation and retrieval operations on objects of the associated classes. They are either implemented in the O-O DBMS or declared by rules which are then processed by a rule processing component of an O-O DBMS. An underlying algebra does not have to distinguish these association types since they are explicitly defined in a schema and their semantics are implemented in an O-O DBMS. For this reason, the algebra introduced for manipulating an O-O database does not have to incorporate the semantics of association types. This is particularly important if the algebra is to be used as a general algebra for supporting a variety of O-O models.

Thus, from the algebra point of view, an O-O semantic database can be viewed as a collection of objects, grouped together in classes, and inter-related among each other through some type-less associations. At the intensional (i.e., schema) level, we can view an O-O database uniformly as a collection of object classes inter-related through type-less associations. For example, a schema for a university database is graphically illustrated in Figure 1. Each rectangle denotes a nonprimitive-class which represents a class containing objects of interest in an application, such as a class of person objects or a class of department objects. Objects in a nonprimitive-class are assigned with system-wide unique object identifiers (or **OID**s). Each circle denotes a primitive-class which represents a class of primitive objects which serve as a domain for defining or describing other objects, e.g., a class of names or a class of age values. Associations among object classes are denoted by the edges between the classes in the schema graph. All edges are bi-directional. Query processing involves the traversal of object classes along these edges.

At the extensional (i.e., instance) level, a database can be viewed as a collection of objects, grouped together in classes, and inter-related through type-less associations. For example, an object graph corresponding to a part of the university schema graph is shown in Figure 2. In this example, the Course object c2 is associated with two Section objects, representing the fact that two sections of the course c2 are offered. The Section object sc2 is taught by Teacher t2 and is taken by Student s1, s2, and s3. Finally, each Student object is associated with a GPA object.

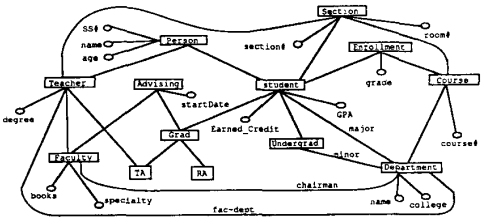

Figure 1 Schema Graph of a University Database

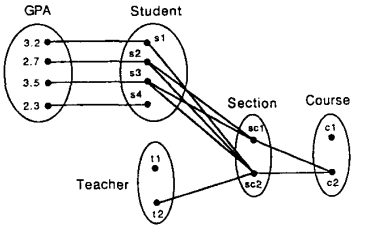

Figure 2 Object Graph

Based on the extensional view of a database, the user can query the database by specifying patterns of object associations as the search condition to select some objects for further processing by either system-defined operations such as retrieval, display, update, insert, delete, etc., or user-defined operations such as rotate a design object, purchase a part, hire a faculty member, etc. For example, the following two queries can be issued against the university database illustrated in Figures 1 and 2.

Query 1: Get the social security numbers of the teaching assistants.

It is clear that objects associated in the following pattern

TA—Grad—Student—Person—SS#

Here, the lines connecting the classes represent the traversal path. They will be replaced by the association operators to be defined later. It should be noted that since the A-algebra is at a level lower than the high-level query languages provided to the user, it provides a way to describe and implement this retrieval by specifying the pattern using some primitive operators rather than including the inheritance concept in the algebra. An association-based high-level language, however, can specify the pattern **TA—SS#** for this query based on the inheritance concept and the query interpreter will translate it into the corresponding A-algebra expression based on the schema definition.

We further note that this and other examples given in this paper assume that objects of a subclass (e.g., **TA**) are also members of its superclasses (e.g., **Student** and **Person**). Instances are the representations of objects in different classes. Thus, "dynamic inheritance" is assumed in this paper which is different from the "static inheritance" of attributes used in SMALLTALK-like O-O models [GOL81]. The A-algebra is applicable to both inheritance methods.

Query 2: For all sections of courses offered by the CIS department, get the specialties of faculty members who are teaching them. Also, get the earned credits and the GPAs of those students who are taking sections of these courses.

To satisfy the query, the following operations are required in the algebra level:

(1) Get the Specialty of the Faculty objects who satisfy the following association pattern,

Specialty—Faculty—Teacher—Section—Course

Name[CIS]—Department ——┘

which represents the specialties of faculty members who are teachers who teaches sections of courses offered by the CIS department.

(2) Get the GPAs and Earned-Credits of the Student objects who satisfy the following association pattern,

GPA—Student—Section—Course—Department—Name[CIS]

Earned_Credit

According to the stated query, the GPAs and Earned_Credits of students who are taking sections of CIS courses which do not have teachers with specialties should be included in the result. Similarly, the specialties of teachers who are teaching sections of CIS courses which do not have students with GPAs and Earned_Credits should also be included. This means that there is a logical OR condition between two subpatterns merged at Section as shown in Figure 3. In this figure, the single arc at Section represents the "OR" condition and the double arc at Student represents the "AND" condition which requires that a student object be associated with both GPA and Earned_Credit.

It should be clear to the reader that this association-based query formulation is quite different from the record-based query formulation in the existing relational query languages which matches key attributes with foreign keys in different relations. A complex pattern of object associations may contain branches with logical AND and OR conditions and complex network structures. Such patterns, which can be specified in an association-based language in a rather straightforward manner, can only be specified in a relational query using complex nested query blocks or multiple queries [ALA89a]. The association algebra to be described below provides the operators and the mathematical basis for processing heterogeneous as well as homogeneous patterns of object associations.

3. Association Algebra (A-algebra)

3.1 Definitions

First, we formally define an O-O database at both schema and object levels.

Schema Graph (the intensional database): The schema graph of an O-O database is defined as **SG(C,A)**, where $C=\{C_i\}$ is a set of vertices representing object classes; **A** is a set of edges, each of which, $A_{i,j}(k)$, represents association between classes C_i and C_j, where **k** is a number for distinguishing the edges from one another when there is more than one edge between two vertices.

Object Graph (the extensional database): The object graph of an O-O database is defined as **OG(O,E)**, where $O=\{O_{i,j}\}$ is a set of vertices representing object instances (jth object instance in class C_i); and $E=\{O_{i,j}\overset{k}{—}O_{m,n}\}$ is a set of edges representing the associations among object instances. When an instance is connected with another in the object graph, a regular-edge (solid line) is drawn between the corresponding vertices as $O_{i,j}\overset{k}{—}O_{m,n}$ which specifies that **j**th instance in class C_i is related to **n**th instance in class C_m through the **k**th association of classes C_i and C_m. If two object instances $O_{i,j}$ and $O_{m,n}$ are not connected in the object graph but their classes C_i and C_m in the corresponding **SG** are directly connected, a complement-edge (dotted line) is drawn between them and is denoted by $O_{i,j}\overset{k}{\cdots}O_{m,n}$.

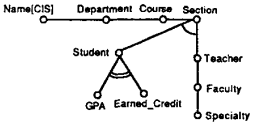

Figure 3 Query by Pattern

In this O-O paradigm, an object may participate in several classes (in a generalization hierarchy/lattice). Its representation in a class is called an object instance.

The reason for explicitly introducing complement-edges into the **OG** is to allow the A-algebra to manipulate both association and non-association between object instances of two adjacent classes. In an O-O database, it is not necessary to explicitly store the complement-edges. Figure 4 illustrates the regular-edges and complement-edges among the object instances of three object classes. For example, we see that section sc1 is taken by students s2 and s3 (regular-edges) and not taken by students s1 and s4 (complement-edges).

To define "association pattern", we first extend the concept of connected graphs in graph theory by treating complement-edges as edges, i.e., a connected graph is a graph in which there exists at least one path between any two vertices and each path may contain regular-edges, complement-edges, or a combination of the two. We shall, from now on, use an upper-case letter to denote a class and the corresponding lower-case letter with a subscript to denote an object instance in that class. We shall assume that there is only one edge between any two vertices in **SG** unless otherwise specified so as not to complicate the notation.

Association Pattern: A connected subgraph of an **OG** is an association pattern (or pattern for short).

By this definition, a single vertex is an association pattern. We call it an Inner-association-pattern (Inner-pattern for short). It is algebraically represented as $(\mathbf{a}_i)$ for a vertex of class **A**. A regular-edge together with two vertices (i.e., two Inner-patterns) it connects is called an Inter-association-pattern (Inter-pattern) which is represented as $(\mathbf{a}_i\mathbf{b}_j)$. This pattern states that object instances $\mathbf{a}_i$ and $\mathbf{b}_j$ are associated with each other in **OG**. A complement-edge together with the two Inner-patterns it connects is called a Complement-association-pattern (Complement-pattern) and is represented as $(\overline{\mathbf{a}_i\mathbf{b}_j})$. This pattern states that $\mathbf{a}_i$ and $\mathbf{b}_j$ are not associated with each other in **OG**. An O-O data contains only Inter-patterns. Complement-patterns are derived during query processing to manipulate object instances that are not associated with one another. If a path consists of only regular-edges between non-adjacent vertices $\mathbf{a}_i$ and $\mathbf{b}_j$, it is represented by a Derived-inter-association-pattern (D-Inter-pattern), denoted by $(\widetilde{\mathbf{a}_i\mathbf{b}_j})$; otherwise, it can be represented by a Derived-complement-association-pattern (D-Complement-pattern), denoted by $(\widetilde{\overline{\mathbf{a}_i\mathbf{b}_j}})$. When a path is represented by a derived pattern, it simply means that two vertices are indirectly (or directly) associated or non-associated but how they are interrelated (the actual path) is of no importance. A D-Inter-pattern is treated as an Inter-pattern and a D-Complement-pattern is treated as a Complement-pattern in the algebraic operations.

The above five types of patterns are the primitive patterns the latter four being binary patterns. Their graphical and algebraic representations are illustrated in Figure 5a. All other connected subgraphs are called complex patterns. For example, the complex pattern shown in Figure 5b1 contains three primitive patterns: two Inter-patterns $(\mathbf{a}_1\mathbf{b}_1)$ and $(\mathbf{b}_1\mathbf{d}_1)$, and a Complement-pattern $(\overline{\mathbf{b}_1\mathbf{c}_1})$. It can be uniquely defined by its algebraic representation as a set of primitive patterns, i.e., $(\mathbf{a}_1\mathbf{b}_1,\mathbf{b}_1\mathbf{d}_1,\overline{\mathbf{b}_1\mathbf{c}_1})$. More examples of complex patterns are shown in Figure 5b.

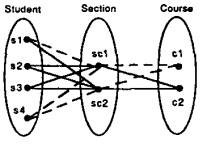

Figure 4 Regular-edges and Complement-edges

The definitions of **OG** and association pattern imply that a pattern is a non-directional graph, i.e., $(\mathbf{a}_i\mathbf{b}_j) = (\mathbf{b}_j\mathbf{a}_i)$, and that the sequence of primitive patterns in the algebraic representation of a complex pattern is not important, hence $(\mathbf{a}_i\overline{\mathbf{b}_j}, \mathbf{b}_j\mathbf{c}_k) = (\mathbf{c}_k\mathbf{b}_j, \mathbf{a}_i\overline{\mathbf{b}_j})$.

Based on the above definition and notion of association pattern, we view an **OG** as an Association Graph (**AG**) and all the association patterns in **AG** form the domain of the A-algebra, denoted as **A**.

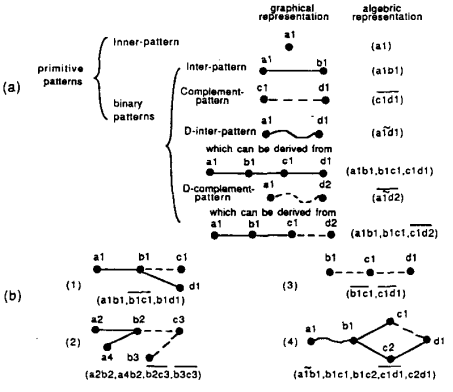

Figure 5 Examples of Association Patterns
(a) primitive patterns (b) complex patterns

3.2 Relationship Between Two Association Patterns

The operators of the A-algebra are defined based on the possible relationships between two patterns in **A**, so that they can be used either to construct complex patterns using simpler patterns or to decompose a complex pattern into several patterns of simpler structures. There are four possible relationships between two patterns $\mathbf{p}^1$ and $\mathbf{p}^2$: non-overlap, overlap, contain, and equal.

(1) **Non-overlap**: Two patterns are said to be non-overlap, denoted by $\mathbf{p}^1 \supset\subset \mathbf{p}^2$, if they have no common Inner-pattern.
(2) **Overlap**: Two patterns are said to be overlapped, denoted by $\mathbf{p}^1 \pi \mathbf{p}^2$, if they have at least one common Inner-pattern.
(3) **Contain**: Contain is a special case of (2) when all the primitive patterns of a pattern ($\mathbf{p}^1$) are contained in another ($\mathbf{p}^2$). We say that $\mathbf{p}^1$ is a subpattern of $\mathbf{p}^2$ and denote this relationship by $\mathbf{p}^1 \subseteq \mathbf{p}^2$.
(4) **Equal**: This is a special case of (3) when a pattern contains all the primitive patterns of another, and vise versa. It is denoted by $\mathbf{p}^1 = \mathbf{p}^2$.

Before defining the nine association operators, we give the definition of "Association-set" -- the operand of the association operators.

Association-set: An association-set, denoted by a Greek letter α (or $\beta,\gamma,\ldots$), is a set of association patterns without duplicates. If a superscript α^i designates the ith pattern in α, then $\alpha^i \neq \alpha^j$ $(i \neq j)$. An empty set is also an association-set, denoted by ϕ.

A special type of association-set is called homogeneous association-set, which is important to the A-algebra, since some of the mathematical properties hold only when operands are homogeneous association-sets.

HomogeneousAssociation–set: An association-set is homogeneous, if

(1) all patterns are formed by Inner-patterns from the same set of object classes; and
(2) all patterns have the same number of Inner-patterns from each class in the set; and
(3) all patterns have the same topology and their corresponding primitive patterns are of the same type.

Otherwise, it is a heterogeneous association-set.

Figure 6 depicts three example association-sets: α is homogeneous, whereas β is not since pattern β^3 has only one Inner-pattern of class **C** instead of two as in β^1 and β^2. γ is not homogeneous because γ^3 contains a Complement-pattern which is different from γ^1 and γ^2.

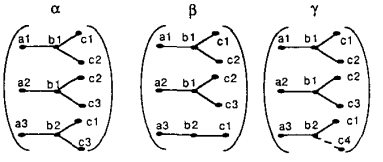

Figure 6 Examples of Association Sets

3.3 Association Operators

Nine association operators are formally defined in this section: two unary operators [A-Project (Π) and A-Select (σ)] and seven binary operators [Association (*), A-Complement (|), A-Union (+), A-Difference (–), A-Divide (÷), NonAssociate (!), and A-Intersect (•)]. The examples used to explain these operators will make use of the domain $\mathcal{A}$ shown in Figure 7. To keep the graph simple, the Complement-patterns are not shown in the figure. The mathematical properties such as commutativity, associativity, and idempotency satisfied by the operators are given after each definition.

3.3.1 Notations

Notations that will be used in the subsequent sections are as follows:

A, B,...,K	Denote classes.
$\mathbf{CL_i}$	Denotes variable for a class.
$\mathbf{[R(CL_1,CL_2)]}$	Denotes the association between classes $\mathbf{CL_1}$ and $\mathbf{CL_2}$.
$\mathbf{a_i}$	Denotes the ith Inner-pattern of class A.
@	Denotes an Inner-pattern variable.
$\mathbf{(a_ib_j)}$	Denotes an Inter-pattern between two classes **A** and **B**.
$\mathbf{(\overline{a_ib_j})}$	Denotes a Complement-pattern between two classes **A** and **B**.
$\mathbf{(\widetilde{a_ic_k})}$	Denotes a Derived-pattern from class **A** to class **C**.
$\alpha, \beta, \gamma,...$	Denote association-sets.
α^i	Denotes ith pattern of association-set α.
{W},{X},{Y},...	Denote sets of classes. Hence, $\alpha_{\{X\}}$ represents α which has Inner-pattern(s) from the classes in **{X}**.

It should be noted that an Inner-pattern (i.e., an object instance) is represented by an instance identifier (**IID**), which is a system-assigned object identifier (**OID**) prefixed by its class identification so that the object instances of an object in multiple classes can be unambiguously distinguished and the fact that these object instances are of the same object can easily be recognized.

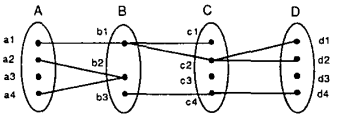

Figure 7 A Sample Database Association Graph
(The C-patterns are not shown)

3.3.2 Operators

(1) Associate (*):

The Associate operator is a binary operator which constructs an association-set of complex patterns by concatenating the patterns of two operand association-sets. Since a pattern may have Inner-patterns from many classes and an object class may have more than one association with another class, it is necessary to specify through which association the concatenation of two patterns is intended. The Associate operation on association-sets α and β over the association **R** between classes **A** and **B** is defined as follows:

$$\alpha * [R(A,B)]\ \beta = \{\ \gamma^k \mid \gamma^k=(\alpha^i,\beta^j,a_mb_n):\ a_mb_n\in[R(A,B)] \wedge a_m\in\alpha^i \wedge b_n\in\beta^j\ \}$$

The result of an Associate operation is an association-set containing no duplicates. Each of its pattern is the concatenation of two patterns (one from each operand association-set). More specifically, if the Inner-pattern (or object instance $\mathbf{a_m}$) of **A** in α^i is associated with the Inner-pattern (or object instance $\mathbf{b_n}$) of **B** in β^j in the domain of the algebra $\mathcal{A}$, then α^i and β^j are concatenated via the primitive pattern $(\mathbf{a_mb_n})$.

An example of the Associate operation is shown in Figure 8a. For clarity, we use graphical notation in the figures. In the example, α^1 is concatenated with β^1 and β^2, respectively, due to the existence of $(\mathbf{b_1c_1})$ and $(\mathbf{b_1c_2})$ in $\mathcal{A}$ as shown in Figure 7. α^2 is dropped simply because it does not have an Inner-pattern of class **B**. α^3 is dropped because $(\mathbf{b_2})$ is not associated with any Inner-pattern of class **C** in $\mathcal{A}$. β^4 cannot be concatenated through $(\mathbf{c_4})$ with any patterns in α because no pattern in α has an Inner-pattern of **B** that is associated with $(\mathbf{c_4})$ in $\mathcal{A}$. For the same reason β^3 is dropped.

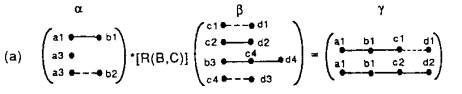

Figure 8a Example of Associate Operation

For the Associate operator, **[R(A,B)]** can be omitted if the following conditions hold: (1) both α and β are A-algebra expressions, (2) the Associate operates on the last class in a linear expression α and the first class in a linear expression β, and (3) there is a unique association between these two classes. For example, **A *[R(A,B)] B** can be written as **A*B**, if class **A** is associated with class **B** through the attribute **[R(A,B)]** of **A**. It should be pointed out that A-algebra allows an attribute have a computed value (or object). For instance, **B=f(A)**. The implementations of the function and the procedure are invisible to the algebra. However, they should not have side effects, i.e., the computed result must be of the same data type as **B**.

The Associate operator is commutative and conditionally associative as defined below:

$$\alpha *[R(A,B)]\ \beta = \beta *[R(B,A)]\ \alpha \qquad \text{(commutativity)}$$

$$(\alpha_{\{X\}} *[R(A,B)]\ \beta_{\{Y\}}) *[R(C,D)]\ \gamma_{\{Z\}} \qquad \text{(associativity)}$$
$$= \alpha_{\{X\}} *[R(A,B)]\ (\beta_{\{Y\}} *[R(C,D)]\ \gamma_{\{Z\}})$$
$$\text{(if } C\notin\{X\} \wedge B\notin\{Z\})$$

The associativity holds true when α and γ do not have Inner-patterns of classes **C** and **B**, respectively. Otherwise, the associativity does not hold. For example, if $\alpha=(a_1b_1,b_1c_2)$, $\beta=(b_1c_1)$, and $\gamma=(d_1)$ and $\mathcal{A}$ shown in Figure 7 is the domain of the algebra, then

$$(\alpha *[R(A,B)]\ \beta) *[R(C,D)]\ \gamma = (a_1b_1,b_1c_1,b_1c_2,c_2d_1)$$

$$\alpha *[R(A,B)]\ (\beta *[R(C,D)]\ \gamma) = \phi$$

(2) A-Complement (|):

The A-Complement operator is a binary operator which concatenates the patterns of two operand association-sets over Complement-patterns. It is used to identify the object instances in two classes which are not associated with each other in $\mathcal{A}$ of Figure 7. The A-Complement operator is defined as follows:

$$\alpha \mid [R(A,B)]\ \beta = \{\ \gamma^k \mid \gamma^k=(\alpha^i,\beta^j,\overline{a_mb_n}):\ (\overline{a_mb_n})\in [R(A,B)] \wedge a_m\in\alpha^i \wedge b_n\in\beta^j$$
$$\text{or}\quad \gamma^k=\alpha^i : \exists (m)(a_m\in\alpha^i) \wedge (\beta=\phi \vee (n)(b_n\in\beta))$$
$$\text{or}\quad \gamma^k=\beta^i : \exists (n)(b_n\in\beta^i) \wedge (\alpha=\phi \vee (m)(a_m\in\alpha))\ \}$$

The result of an A-Complement operation is an association-set. Each of its patterns is formed by concatenating two patterns (one from each operand association-set) via a Complement-pattern ($\overline{a_mb_n}$), where a_m and b_n belong to α^i and β^j, respectively, and the Complement-pattern ($\overline{a_mb_n}$) is in $\mathcal{A}$. In the special case when α(**or** β) is an empty association-set or does not have Inner-patterns of **A**(**or B**), then all patterns of β(**or** α) that have Inner-patterns of **A**(**or B**) are retained in the resulting association-set.

An example of the A-Complement operation is shown in Figure 8b. It operates over the association between classes **B** and **C**. α^2 does not appear in the resultant association-set because it contains no Inner-patterns of **B**. α^1 cannot be A-Complemented with β^1 and β^2 because it is connected with β^1 and β^2 by Inter-patterns (b_1c_1) and (b_1c_2) in $\mathcal{A}$, respectively.

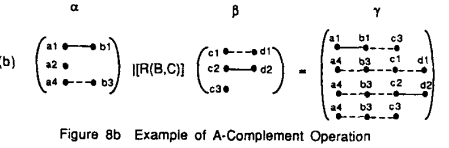

Figure 8b Example of A-Complement Operation

Under the same conditions as given in the Associate operator, [**R**(**A**,**B**)] need not be specified with the A-Complement operator unless there is an ambiguity. The A-Complement operator is commutative and associative. For the similar reason described for the Associate operator, the associativity holds true conditionally.

$$\alpha \mid [R(A,B)]\ \beta = \beta \mid [R(B,A)]\ \alpha \qquad \text{(commutativity)}$$

$$(\alpha_{\{X\}} \mid [R(A,B)]\ \beta_{\{Y\}}) \mid [R(C,D)]\ \gamma_{\{Z\}} \qquad \text{(associativity)}$$
$$= \alpha_{\{X\}} \mid [R(A,B)]\ (\beta_{\{Y\}} \mid [R(C,D)]\ \gamma_{\{Z\}})$$
$$(\text{if } C\notin\{X\} \wedge B\notin\{Z\})$$

(3) A-Select (σ)

The A-Select is a unary operator, which operates on the association-set α to produce a subset of patterns that satisfy a specified predicate **P**. A pattern in the operand association-set is retained iff the predicate is evaluated true for that pattern.

$$\sigma(\alpha)[P] = \{\ \gamma \mid \gamma^i = \alpha^i : P(\alpha^i)=\textbf{true}\ \}$$

where α is defined by an algebraic expression, and $\mathbf{P} = T_1\theta_1T_2\theta_2\cdots\theta_{n-1}T_n$. Each term, T_i(i=1,2,...n), is a comparison between two expressions and θ_i(i=1,2,...,n−1) is a Boolean operator ($\wedge$ or $\vee$). $P(\alpha^i)$=**true** represents that a pattern is evaluated true for that predicate.

The expressions on the left- and right-hand sides of a comparison operation may contain constants, functions and/or operations on object instances, but cannot both be constants. The comparison terms are type sensitive, i.e., the results of the two expressions in a term must be data of the same type for primitive-classes or both **IID**s for nonprimitive-classes. $=,>,<,\geq,\leq$, and $\neq$ are the legitimate comparisons for numerical types; = and $\neq$ for character, string, and **IID** types; and $=,\subset,\supset,\subseteq,\supseteq$, and $\neq$ for set types. The comparison of two **IID**s is performed by comparing their **OID** portions, since **IID**s are the concatenation of the class identifiers and **OID**s. A single valued object or a single **IID** can be treated either as its own data type in numerical, string, or **IID** comparison, or as a set type containing one element in set comparison.

As an example of A-Select, we assume that there are two associated classes: S for stack and Q for queue. To select associated stack and queue object pairs in which the object on the top of the stack is located in the front or the tail of the queue, the algebraic expression can be written as

$$\sigma(S*Q)[\text{top}(S) \subset (\text{front}(Q) \cup \text{tail}(Q))]$$

where $\cup$ is a set-union operation. For the top equals the front and the bottom equals the tail, we have

$$\sigma(S*Q)[(\text{top}(S)=\text{front}(Q)) \wedge (\text{bottom}(S)=\text{tail}(Q))]$$

(4) A-Project (Π):

Similar to the projection operation in the relational algebra, an A-Project operation projects a given pattern over some subpattern(s). However, in relational algebra, the relationship among the projected attributes is not important. Whereas in A-algebra, the associations among the projected subpatterns must be maintained so that the associations among the object instances in these subpatterns will be retained. The A-Project operator is defined as follows:

$$\Pi(\alpha)[E;\ T]$$

where α is an association-set defined by an A-algebra expression; $E=(e_1, e_2, \ldots, e_n)$ is a set of expressions which specify subpatterns to be projected; and $T=(t_1,t_2, \ldots, t_m)$ is a set of ordered sets of classes. Each ordered set, t_i, defines a path for connecting two projected subpatterns specified by the **E** expressions. It contains a minimal number of classes along the path which can uniquely identify that path.

The result of an A-Project is an association-set which contains object association patterns corresponding to the projected subexpressions specified in **E**. Each pair of subpatterns is connected by either a D-inter-pattern or a D-complement-pattern depending on whether the path connecting these two subpatterns contains all Inter-patterns or not. The function of D-inter-pattern and Derived-complement-pattern is to retain the original associations among object instances before the projection operation.

Figure 8c shows an example of A-Project from a pattern α over **A***B and **D**. For α^1, the subpatterns (a_1b_1) and (d_1) satisfy **A***B and **D**, respectively. Therefore, they are kept in the result. According to the path specification, a D-Complement-pattern ($\overline{b_1d_1}$) is added to the result, thus $\gamma^1=(a_1b_1, d_1, \overline{b_1d_1})$, i.e., $\gamma^1=(a_1b_1, \overline{b_1d_1})$. γ^2 is produced for the same reason. Since α^3 does not have a subpattern satisfying **A***B, only (d_3) is retained.

Figure 8c Example of A-Project Operation

(5) NonAssociate (!):

The NonAssociate operator is a binary operator used to identify the association patterns in one operand association-set that are not associated (over a specified association) with any pattern in the other association-set, and vise versa, based on the patterns in A. The NonAssociate operator is defined as follows:

$$\alpha\ !\ [R(A,B)]\ \beta = \{\ \gamma \mid$$
$$\gamma^k = (\alpha^i, \beta^j, \overline{a_m b_n}):\ (\overline{a_m b_n}) \in [R(A,B)]\ \wedge\ a_m \in \alpha^i\ \wedge\ b_n \in \beta^j$$
$$\wedge\forall\ ((a_m b_{n'}), (a_{m'} b_n) \in A)(a_{m'} \notin \alpha\ \wedge\ b_{n'} \notin \beta)$$
$$\text{or}\ \ \gamma^k = \alpha^i:\ \exists (m)(a_m \in \alpha^i)\ \wedge \nexists (n)(b_n \in \beta)$$
$$\forall\forall\ (b_n \in \beta)\ \exists\ (p,\ p \neq m)(a_p \in \alpha\ \wedge\ (a_p b_n) \in [R(A,B)])$$
$$\text{or}\ \ \gamma^k = \beta^j:\ \exists (n)(b_n \in \beta^j) \wedge \nexists\ (m)(a_m \in \alpha)$$
$$\forall\forall\ (a_m \in \alpha)\ \exists\ (p,\ p \neq n)(b_p \in \beta\ \wedge\ (a_m b_p) \in [R(A,B)])\ \}$$

The result of a NonAssociation operation is an association-set. Each of its patterns is formed by concatenating two patterns α^i and β^j via a Complement-pattern $(\overline{a_m b_n})$ under the condition that α^i is not associated with any β^j and vice versa. Furthermore, in the special case where the patterns of α(or β) have Inner-patterns of **A**(or **B**) and cannot be concatenated with any patterns of β(or α), then these patterns of α(or β) will be retained in the result if one of the following three conditions holds: (1) β(or α) is an empty association-set, (2) all patterns of β(or α) do not have Inner-patterns of **B**(or **A**), or (3) all patterns of β(or α) that have Inner-patterns of **B**(or **A**) can be concatenated with patterns of α(or β).

An example of the NonAssociate operation is shown in Figure 8d. In the example, α^1 and β^1 are dropped due to the existence of (b_1c_2) in Figure 7. α^2 is dropped because it does not contain an Inner-pattern of class **B**. β^3 is dropped because it does not contain an Inner-pattern of class **C**. γ^1 is in the resultant association-set because (b_2) is not associated with (c_4) in $\mathcal{A}$ shown in Figure 7 and none other pattern in α is associated with (c_4). γ^2 exists because (b_2) is not associated with (c_3) in $\mathcal{A}$.

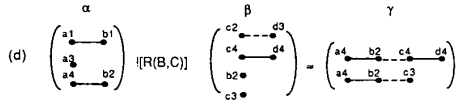

Figure 8d Example of NonAssociate Operation

Note that the NonAssociate operator produces a resultant association-set which is a subset of that produced by the A-Complement operator. NonAssociate is not a primitive operator since it can be expressed by other operators. However, it is very useful for query formulation and is therefore included in the set of A-algebra operators.

Under the same conditions as given in the Associate operator, **[R(A,B)]** need not be specified unless there is an ambiguity. The NonAssociate operator is commutative but not associative.

$$\alpha\ !\ [R(A,B)]\ \beta = \beta\ !\ [R(B,A)]\ \alpha \qquad \text{(commutativity)}$$

(6) A-Intersect (•):

The A-Intersect operation is convenient for constructing a pattern with a branch, a lattice, or a network structure, since a pattern in such a structure can be viewed as the intersection of two patterns. Conceptually, the A-Intersect operator is equivalent to the JOIN operator in the relational algebra. It operates on two operand association-sets over a set of specified classes. Two patterns, one from each association-set, are combined into one if they contain the same set of Inner-patterns for each specified class.

$$\alpha_{\{X\}} \bullet\{W\}\ \beta_{\{Y\}} = \{\ \gamma \mid\ \gamma^k = (\alpha^i, \beta^j):$$
$$\forall\ (CL_n \in \{W\})\ \forall\ (@ \in CL_n, \alpha^i)(@ \in \beta^j)$$
$$\wedge\forall\ (CL_n \in \{W\})\ \forall\ (@ \in CL_n, \beta^j)(@ \in \alpha^i)\ \}$$

Figure 8e shows an example of the A-Intersect operation over classes **B** and **C**. The resultant association-set contains four patterns, which are the intersection of $\alpha^1 \pi \beta^1$, $\alpha^1 \pi \beta^2$, $\alpha^2 \pi \beta^1$, and $\alpha^2 \pi \beta^2$, respectively, since they all have Inner-patterns (b_1) and (c_2). Other patterns $(\alpha^3, \alpha^4, \beta^3, \beta^4)$ fail to produce new patterns because they either have no Inner-pattern in both classes **B** and **C** or have no common Inner-pattern of class **C**.

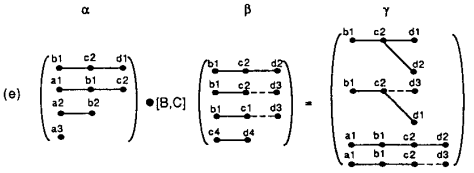

Figure 8e Example of A-Intersect Operation

The set of classes $\{W\}$ can be omitted when the A-Intersect operation is performed on all the common classes of its operands, i.e., $\{W\}=\{X\}\cap\{Y\}$ is implied.

Since a network structured pattern can be transformed into a set of other simple patterns, an A-Intersect operation for building a complex pattern can be replaced by an Associate operation followed by an A-Select operation. The A-Intersect operator is commutative, conditionally associative and idempotent.

$$\alpha \bullet\{W\}\ \beta = \beta \bullet\{W\}\ \alpha \qquad \text{(commutativity)}$$

$$(\alpha_{\{X\}} \bullet\{W_1\}\ \beta_{\{Y\}}) \bullet\{W_2\}\ \gamma_{\{Z\}} \qquad \text{(associativity)}$$
$$= \alpha_{\{X\}} \bullet \{W_1\}\ (\beta_{\{Y\}} \bullet\{W_2\}\ \gamma_{\{Z\}})$$
$$(\text{if}\ \ (\{W_1\}-\{W_2\}) \cap \{Z\} = \phi\ \wedge\ (\{W_2\}-\{W_1\}) \cap \{X\} = \phi)$$

$$\alpha \bullet \alpha = \alpha \qquad \text{(idempotency)}$$

(if α is a homogeneous association-set)

The associativity is not always true because there are cases in which a pattern of β that fails to intersect with any patterns of γ, may succeed by first intersecting with a pattern of α in the operation $(\bullet\{W_1\})$ and then intersecting with a pattern of γ in the operation $(\bullet\{W_2\})$.

Now we define three set operators, which are different from the corresponding set operators in relational algebra, since they operate on heterogeneous structures rather than union-compatible relations.

(7) A-Union(+):

Similar to the UNION operation of the relational algebra, A-Union combines two association-sets into one. However, these two association-sets may contain heterogeneous association structures. It is important for A-algebra to be able to operate on heterogeneous structures because some prior operations may produce heterogeneous association-sets and may need to be further processed over the object instances of a common class against other patterns of associations. Unlike the relational algebra and other O-O query languages, union-compatibility is not a restriction in A-algebra. For this reason, A-algebra has more expressive power. Any query that can be expressed by a single expression in other languages can be expressed as a single A-algebra expression but not vise versa. The A-Union operation is defined as follows:

$$\alpha + \beta = \{\ \gamma \mid \gamma^k \in \alpha \vee \gamma^k \in \beta\ \}$$

The A-Union operator is commutative, associative, and idempotent:

$$\alpha + \beta = \beta + \alpha \qquad \text{(commutativity)}$$

$$(\alpha + \beta) + \gamma = \alpha + (\beta + \gamma) \qquad \text{(associativity)}$$

$$\alpha + \alpha = \alpha \qquad \text{(idempotency)}$$

(8) A-Difference (–):

The A-Difference implements the same concept as the DIFFERENCE operator in relational algebra but with two differences. First, its operands do not have to be union-compatible. Secondly, a pattern in the minuend is retained if it does not contain any of the patterns in the subtrahend.

$$\alpha - \beta = \{\ \gamma \mid \gamma^k = \alpha^i : \nexists (\beta^j)(\beta^j \subseteq \alpha^i)\ \}$$

The example depicted in Figure 8f shows that α^1 and α^3 are dropped since they both contain β^1.

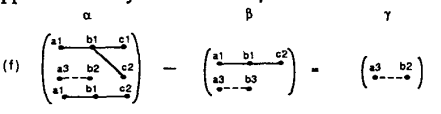

Figure 8f Example of A-Difference Operation

(9) A-Divide (÷)

The A-Divide operator implements the concept that a group of patterns with certain common features contains another set of patterns.

$$\alpha \div_{\{W\}} \beta = \{\ \gamma \mid \gamma^k = \alpha_s^i : \forall\ (j)(\beta^j \subseteq \alpha_s^i)\ \}$$

where α_s is a subset of patterns of α, which have common Inner-patterns for all classes of {W}. If {W} is not specified, the A-Divide operation retains all the patterns of α, each of which contain at least one pattern of β and they together contain all patterns of β.

Figure 8g shows an example of α being divided by β with respect to class **B**. The A-Divide operation retains α^1, α^2 ,and α^3 since they all contain Inner-pattern (b_1) of **B** and together contain all patterns of β.

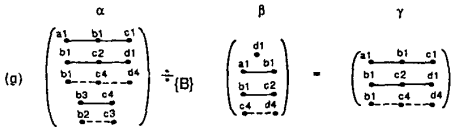

Figure 8g Example of A-Divide Operation

3.3.3 Precedence

The precedence relationships of the above operators are as follows. Unary operators have higher precedence than binary operators. The precedence of the seven binary association operators is given in the following order: *, |, !, •, /, –, and +. Parentheses can be used to alter the precedence relationships.

3.3.4 Query Examples

We have formally defined nine association operators and given their simple mathematical properties. Before exploring other properties, we give some examples to illustrate how these operators are used to express queries for processing an O-O database. There can be many alternative expressions for the same query. Choosing the best one for execution is the task of a query optimizer. The mathematical properties of these operators can be used for that purpose.

It is straightforward to write an algebraic expression for a subgraph with a linear structure. The expression for Query 1 given in Section 2 can be formulated as follows by navigating the schema graph (Figure 1) from class **TA** up to class **SS#**.

Π(TA*GRAD*Student*Person*SS#)[SS#]

Shown below is the expression for Query 2 (Section 2). The expression for this query can be easily constructed by specifying the α expression (defining the operand association-set of A-Project) first, and then attaching the [E; T] clause.

Π (σ(Name)[Name="CIS"]*Department*Course*(Section *Teacher*Faculty*Speciality+Section*(Student*GPA•Student *Earned_Credit))) [Section,Specialty,GPA,Earned_Credit; Section:Specialty,Section:GPA,Section:Earned_Credit]

To write this expression, the user may first trace the Schema Graph to specify a graph as shown in Figure 3. Then label each edge of the graph with an operator *, |, or ! depending on the semantics of the query. For this query, every edge is labeled by an *. Since the query does not require that Student and Teacher relate to the same Section of a Course, the two branches should be A-Unioned for further processing. The A-Select operation ensures that only the CIS department is under consideration. The result of this expression will contain two types of patterns: **Section—Specialty** and **GPA—Section—Earned_Credit** as specifies by the [E, T] clause of the A-Project operation. Note that this query cannot be phrased in a single relational algebraic expression, since the union of heterogeneous structures is involved.

Some other query examples are given below. Their corresponding patterns are depicted in Figure 9. The interpretations of these queries are left to the reader.

Query 3: List the names of students who teach in the same departments as their major departments.

Π(Student*Person*Name•Student*Department •Student*Grad*TA*Teacher*Department)[Name]

Query 4: List the section# of those sections which have not been assigned a room or a teacher.

Π(Section#*(Section!Room#+Section!Teacher))[Section#]

Query 5: List the names of students who take courses 6010 and 6020.

Π(Name*Person*Student*Enrollment*Course*Course# +[Student] σ(Course#)[Course#=6010 or Course#=6020])[Name]

Again, we stress that the above association pattern expressions represent the internal algebraic operations that need to be performed if the dynamic inheritance mothod is used. The highlevel query statements that the user uses can be much simpler due to the inheritance of attributes in the generalization hierarchy or lattice.

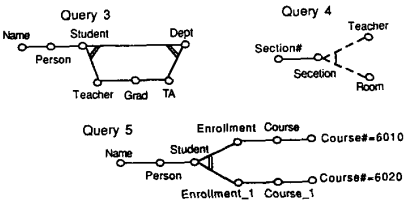

Figure 9 Patterns for Query 3, 4, 5

4. Additional Properties of the Operators

In Section 3.3, we have shown some properties of the individual operators. Many other properties have been studied. These properties are important for query decomposition and query optimization. Due to space limitation, we shall only give the distributivity properties of some operators with respect to other operators.

a) distributive property of * with respect to +:

$$\alpha *[R(A,B)] (\beta + \gamma) = \alpha *[R(A,B)] \beta + \alpha *[R(A,B)] \gamma$$

b) distributive property of | with respect to +:

$$\alpha | [R(A,B)] (\beta + \gamma) = \alpha | [R(A,B)] \beta + \alpha | [R(A,B)] \gamma$$

c) distributive property of • with respect to + :

$$\alpha \bullet\{X\} (\beta + \gamma) = \alpha \bullet\{X\} \beta + \alpha \bullet\{X\} \gamma$$

These three properties hold true for the same reasons. First, the A-Union operation simply lumps together patterns of two association-sets without modifying them. Second, for operation *, |, or •, the decision whether a new pattern is produced or not is determined only based on the structure of the two patterns being operated on and is independent of the other patterns in the operand association-sets.

d) distributive property of * with respect to •:

$$\alpha_{\{X\}} *[R(CL_1,CL_2)] (\beta_{\{Y\}} \bullet\{W\} \gamma_{\{Z\}})$$

$$= \alpha_{\{X\}} *[R(CL_1,CL_2)] \beta_{\{Y\}} \bullet\{W \cup X\} \alpha_{\{X\}} *[R(CL_1,CL_2)] \gamma_{\{Z\}}$$

e) distributive property of | with respect to •:

$$\alpha_{\{X\}} | [R(CL_1,CL_2)] (\beta_{\{Y\}} \bullet\{W\} \gamma_{\{Z\}})$$

$$= \alpha_{\{X\}} | [R(CL_1,CL_2)] \beta_{\{Y\}} \bullet\{W \cup X\} \alpha_{\{X\}} | [R(CL_1,CL_2)] \gamma_{\{Z\}}$$

f) distributive property of ! with respect to •:

$$\alpha_{\{X\}} ![R(CL_1,CL_2)] (\beta_{\{Y\}} \bullet\{W\} \gamma_{\{Z\}})$$

$$= \alpha_{\{X\}} ![R(CL_1,CL_2)] \beta_{\{Y\}} \bullet\{W \cup X\} \alpha_{\{X\}} ![R(CL_1,CL_2)] \gamma_{\{Z\}}$$

Distributive properties d), e), and f) hold true under the following three conditions:

i) $CL_2 \in W$, and
ii) $X \cap Y = X \cap Z = \phi$, and
iii) α is a homogeneous association-set.

The first condition ensures that the *, |, and ! operations are performed on the intersection of β^I and γ^I; otherwise, it does not make sense to have an operation between α and γ. The second condition states that patterns of α are non-overlapping with patterns of β and γ. The third condition states that, on the right-hand side of the expression, only the patterns having the same patterns of α as their subpatterns will succeed in the A-Intersect operation.

The above mathematical properties are the basis for query optimization since they provide ways for transforming a query expression into alternative expressions which produce the same result but with different performances. For example, the associativity holds true for the two Associate operations in the expression **A * B * C**. Thus, either order of evaluation will produce the same result. Similar to the optimization of consecutive JOIN operations in relational databases, the order of their execution should depend on their selectivities (similar to the selectivity of a JOIN operation).

As a second example, the following expression defines a complex pattern as shown in Figure 10a.

$$\mathbf{expr = A * (B*E*F + B * (C*D*H \bullet C*G))}$$

By applying the distributivity of the operators, it can be written as below:

$$\mathbf{expr = A * (B*E*F + B*C*D*H \bullet B*C*G)}$$

$$\mathbf{= A*B*E*F + A*(B*C*D*H \bullet B*C*G)}$$

$$\mathbf{= A*B*E*F + A*B*C*D*H \bullet A*B*C*G}$$

The final expression and the intermediate expressions are the alternatives for evaluation. Among them, the final expression is particularly suitable for a parallel system, since it is an A-Union of two sub-expressions, each of which can be evaluated independently and produces a homogeneous association-set with simpler structure (see Figure 10b). Its processing will be more efficient than the processing over heterogeneous association-set. The second sub-expression can be further optimized using other properties of its operators.

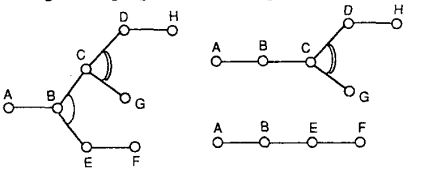

Figure 10 Example of Query Optimization

5. Conclusion

The mathematical foundation of the relational model, namely the relational algebra and the relational calculus, has contributed to the success and popularity of the relational model and relational database management systems. Although the recently introduced O-O DBMSs and their underlying models exhibit several desirable features that are suitable for modeling and processing complex objects found in more advanced database applications, they still do not have a solid mathematical foundation. Such a foundation is important for the efficient manipulation of O-O databases and for the design of high-level query languages to ease the user's task in accessing and manipulating O-O databases.

In this paper, we have presented an algebra for O-O database processing. In this algebra, objects (object instances) and their associations in an O-O database are uniformly represented by association patterns. Nine algebraic operators have been introduced for manipulating patterns of heterogeneous and homogeneous structures. The result of performing an A-algebra expression is an association-set. The closure property of the algebra allows the result of an algebraic expression to be further processed by the algebra.

Several mathematical properties of the A-algebra operators have been described. The A-algebra is complete in the sense that all possible subdatabases that are derivable from an O-O database can be expressed in terms of A-algebra expressions. The proof of completeness is done by induction on the number of classes in SG. More mathematical properties, their proofs, and the completeness issue are presented in a forthcoming paper [SU90].

REFERENCES

[ALA89a] Alashqur, A.M., "A Query Model and Query and Knowledge Definition Languages for Object-oriented Databases," doctoral dissertation, University of Florida, 1989.

[ALA89b] Alashqur, A.M., Su, S.Y.W., and Lam, H., "OQL: A Query Language for Manipulating Object-oriented Databases," Proceedings of the 5th Intl. Conference on VLDB, Amsterdam, The Netherlands, 1989, pp. 433-442.

[ALA90] Alashqur, A.M., Su, S.Y.W., and Lam, H., "A Rule-based Language for Deductive Object-Oriented Databases," Proceedings of the 6th International Conference on Data Engineering, Los Angeles, CA, Feb. 5-9, 1990.

[BAN87] Bancilhon, F., et. al., "FAD, a Powerful and Simple Database Language," Proceedings of the 13th VLDB Conference, Brighton, 1987, pp. 97-105.

[BAN88] Banerjee, J., et. al., "Queries in Object-oriented Databases," Proceedings of the 4th Intl. Conference on Data Engineering, Los Angeles, CA, 1988, pp. 31-38.

[BAT84] Batory, D.S. and Buchmann, A.P., "Molecular Objects, Abstract, Abstract Data Types and Data Models: A Framework," Proceedings Intl. Conference on VLDB, 1984, pp. 172-184.

[CAR88] Carey, M. J., et. al., "A Data Model and Query Language for EXODUS," ACM-SIGMOD Conference 1988, pp. 413-423.

[CHU90] Chuang, H. S., "Operational Role Processing in a Prototype OSAM* KBMS," Master's thesis, University of Florida, 1990.

[COD70] Codd, E., "A Relational Model of Data for Large Shared Data Bank," CACM Vol. 13, No. 6, 1970, pp. 377-387.

[COD72] Codd, E., "Relational Completeness of Database Sublanguages," in Data Base Systems, Rustin, R. (ed.), Prentice-Hall Inc., Englewook Cliffs, NJ, 1971, pp.65-98.

[COL89] Colby, L. S. "A Recursive Algebra and Query Optimization for Nested Relations," ACM-SIGMOD Conference 1989, pp. 273-283.

[FIS87] Fishman, D. H., et. al., "Iris: An Object-Oriented Database Management System," ACM Transaction on Office Information Systems, 5:1, 1987, pp49-69.

[GOL81] Goldberg, A., "Introducing the Smalltalk-80 System," Byte, Aug. 1981, pp.14-26.

[HAM81] Hammer, M. and Mcleod, D., "Database Description with SDM: A Semantic Association Model," ACM TODS, Vol. 6, No. 3, 1981, pp. 351-368.

[KIM87] Kim, W., et. al., "Composite Object Support in an Object-oriented Database System," Proceedings of OOPSLA, Oct. 4-8, 1987, FL, pp. 118-125.

[KIN84] King, R., "Sembase: A Semantic DBMS," the Proceedings of the First International Workshop on Expert Database Systems," Oct. 1984, pp.151-171.

[LAM89] Lam, H., et. al., "Prototype Implementation of an Object-oriented Knowledge Base Management System," Proceedings of PROCIEM '89, Orlando, FL., Nov. 13-15, 1989.

[LEC88] Lecluse, C., Richard, P., and Velez, F., "O_2, an Object-Oriented Data Model," ACM-SIGMOD Conference 1988, pp. 425-433.

[MAC85] MacGregor, R., "ARIEL--A Semantic Front-End to Relational DBMSs," Proceedings of VLDB 85, Atlanta, GA., April 1988.

[MAI86] Maier, D., et. al., "Development of an Object-oriented DBMS," Proc. of OOPSLA '86 Conference, Sept. 29 - Oct. 2, Portland, Oregon, pp. 472-482.

[MAN86] Manola, F. and Dayal, U., "PDM: An Object-Oriented Model," Int'l Workshop On Object-Oriented Database Systems, 1986, pp 18-25.

[PAN89] Pant, S., "An Intelligent Schema Design Tool for OSAM*," Master's thesis, University of Florida, 1990.

[ROW87] Rowe, L. A and Stonebraker, M. R., "The POSTGRES Data Model," Proceedings of the 13th VLDB Conference, Brighton 1987, pp. 83-96.

[SHA90] Shaw G. M., and Zdonic, S. B., "A Query Algebra for Object-Oriented Databases," IEEE Trans. on Data Engineering, 12(3), pp. 154-162, Feb. 1990.

[SHI81] Shipman, D., "The Functional Data Model and the Data Language DAPLAX," ACM Trans. Database System 6(1), March 1918.

[SIN90] Singh M., "Transaction Oriented Rule Processing in an Object-Oriented Knowledge Base Management System," Master's thesis, University of Florida, 1990.

[SU86] Su, S.Y.W., "Modeling Integrated Manufacturing Data With SAM*," IEEE Computer, January, 1986, pp.34-49.

[SU88] Su, S.Y.W., et. al., "An Object-oriented Computing Environment for Productivity Improvement in Automated Design and Manufacturing: Project Summary," PROCIEM '88,Orlando, FL., Nov. 14-15, 1988.

[SU89] Su, S.Y.W., Krishnamurthy, V., and Lam, H, "An Object-oriented Semantic Association Model (OSAM*)," AI in Industrial Engineering and Manufacturing: Theoretical Issues and Applications, Kumara, S., Soyster, A.L., and Kashyap, R.L. (eds.), American Institute of Industrial Engineering, 1989.

[SU90] Su, S.Y.W., Guo, M., and Lam, H., "Association Algebra: A Mathematical Foundation for Object-Oriented Databases," to be submitted to Trans. on Knowledge and Data Engineering.

[TSU84] Tsurt, S. and Zaniolo, C., "An Implementation of GEM -- Supporting a Semantic Data Model on a Relational Back End," Proceedings of the ACM SIGMOD Intl. Conference on the Management of Data, 1984, pp. 286-295.

[TY88] Frederick Ty, "The Design and Implementation of a Graphics Interface for an Object-oriented Language," Master's thesis, University of Florida, 1988.

[ZAN83] Zaniolo, C., "The Database language GEM," Proceedings of the ACM SIGMOD Intl. Conference on the Management of Data, 1983.

[ZDO86] Zdonik, S. B., Skarra, A. H., and Reiss, S. P., "An Object Server for an Object-oriented Database System," International Workshop on Object-oriented Database Systems, Pacific Grove, CA., Sept. 1986.

[ZDO87] Zdonik, S.B., "The implementation of a Shared, Clustered Memory System for an O-O Database System," ACM Trans. on Office Information Systems, Apr. 1987.