SEMANTIC QUERY OPTIMIZATION IN AN OBJECT-ORIENTED SEMANTIC ASSOCIATION MODEL (OSAM*)

By

GARRETT L. GLEASON

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

1990

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFIT/CI/CIA- 89-182
6a. NAME OF PERFORMING ORGANIZATION AFIT STUDENT AT UNIV OF FLORIDA 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION AFIT/CIA
6c. ADDRESS (City, State, and ZIP Code)	7b. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB OH 45433-6583
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification) (UNCLASSIFIED)

SEMANTIC QUERY OPTIMIZATION IN AN OFJECT-ORIENTED SEMANTIC ASSOCIATION MODEL(OSAM*)

12. PERSONAL AUTHOR(S)
GARRETT L. GLEASON

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
THESIS/DISSERTATION	FROM _____ TO _____	1989	117

16. SUPPLEMENTARY NOTATION
APPROVED FOR PUBLIC RELEASE IAW AFR 190-1
ERNEST A. HAYGOOD, 1st Lt, USAF
Executive Officer, Civilian Institution Programs

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL ERNEST A. HAYGOOD, 1st Lt, USAF	22b. TELEPHONE (Include Area Code) (513) 255-2259 22c. OFFICE SYMBOL AFIT/CI

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

AFIT/CI "OVERPRINT"

To my parents,
Walter and Iona.

ACKNOWLEDGEMENTS

I wish to thank Dr. Stanley Su for his never-ending support, guidance, and encouragement throughout my research. I am also grateful to Dr. Herman Lam for his thoughtful insights and help through our countless meetings. Finally, I appreciate Dr. Randy Chow's willingness to participate on my committee and his comments on my work.

I also wish to recognize Sharon Grant, our over-worked but much appreciated secretary, for the help she has given me. I am thankful to all the knowledgeable and helpful students of the Database Center for their guidance and thought provoking discussions. Their help was invaluable.

This research was supported by the United States Air Force through the Air Force Institute of Technology's graduate program and by the National Science Foundation, Grant Number DDN-A814989.

TABLE OF CONTENTS

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

SEMANTIC QUERY OPTIMIZATION IN AN
OBJECT-ORIENTED SEMANTIC ASSOCIATION MODEL (OSAM*)

By

Garrett L. Gleason

May 1990

Chairman: Dr. Stanley Y. W. Su
Major Department: Computer and Information Sciences

The Object-oriented Semantic Association Model (OSAM*) represents the semantics of a knowledge base in terms of structural properties, operational characteristics, and knowledge rules associated with the objects of concern in an application. Because of its rich semantic modeling capabilities, OSAM* can be used as the underlying model for constructing a more powerful knowledge base management system (KBMS) than those database management systems that use the traditional relational data model. Knowledge rules in a KBMS serve to uphold the semantic properties and security/integrity constraints and to derive data that is not explicitly stored in the knowledge base. The knowledge rules in OSAM*'s KBMS can be specified in two ways: (1) frequent integrity constraints

over

are structurally specified by key-words in OSAM*'s Semantic-diagram (S-diagram) and (2) user defined constraints and deductive rules are specified by a rule specification language. The knowledge rules are important sources of information for query optimization (i.e., semantic query optimization). This thesis (1) analyzes the integrity constraints and deductive rules (knowledge rules) allowable in OSAM*, (2) defines a set of transformation rules which can be used for query optimization, (3) uses these transformation rules in a number of case studies to show how query plans can be generated, and (4) discusses the cost estimation for evaluating the query plans.

CHAPTER 1
INTRODUCTION

In database management technology, there is a trend toward building a database management system (DBMS) with higher functionality and ease of use. The higher functionality is achieved by using semantic models that capture the semantics of an application in terms of structural properties, the operational characteristics, and the knowledge rules. High-level models and non-procedural languages of knowledge base management systems (KBMS) have contributed greatly to ease-of-use. Very complex query requests are easier to formulate by allowing the user to specify what information is desired, rather than a step-by-step specification of how to get that desired information.

Efficiency becomes a key issue in the implementation of a DBMS/KBMS that uses a semantic model and a high-level query language. The concern of employing efficient processing no longer rest on the user but rather within the DBMS/KBMS itself. Query optimization techniques, which make use of the rich semantic information captured by a semantic model, are important to make the intelligent DBMS/KBMS perform efficiently. The knowledge rules of a KBMS that serve to uphold semantic properties and security/integrity constraints and to

derive data that is not explicitly stored in the knowledge base (deductive rules), can also serve as a basis for optimizing high-level queries. As important sources of information, knowledge rules can be employed to generate a series of alternative query execution plans that are projected to be more efficient than evaluating the query in its original form. The multiple query execution plans generated represent semantic equivalent queries (i.e., queries that will produce the same result provided the knowledge base is in a valid state).

There are many existing works on query optimization [GIR78], [HAM80], [KIN81], [JAR84a], [JAR84b], [CHA85], [FRE87], [GOE87], [LOB88], and [GOE89]. Most of them are based in the relational data model which captures a limited amount of structural properties and integrity constraints. Relational-data-model-based KBMSs employ query and rule specification languages that are based on attribute values (i.e., queries and rules are defined in terms of attributes of relations). As such, the languages weakly support (if at all) queries and rules based on semantic relationships between objects. Semantic query optimization in these KBMSs. in turn, cannot take full advantage of known semantic associations between objects.

In this work, we use the Object-oriented Semantic Association Model (OSAM*) [SU86] and [SU89] as an example semantic model to show that constraints specified structurally and user-defined rules can be used for query

optimization. This model's KBMS employs query and rule specification languages that are based on associations between data objects. The languages explicitly define queries and rules in terms of patterns of object associations. Complex queries and rules can therefore be easily formulated. In this manner, the knowledge rules in OSAM*'s KBMS capture more semantic properties than the relational-data-model-based rule specification languages. Thus, semantic query optimization is also developed on the basis of associations among data objects. This thesis analyzes the integrity constraints and deductive rules (knowledge rules) allowable in OSAM*, defines a set of transformation rules to be used for query optimization, uses these transformation rules in a number of case studies to show how alternative query plans can be generated, and discusses the cost estimation for evaluating the query plans.

This thesis is organized as follows: Chapter 2 introduces a KBMS designed on the basis of the OSAM*, including the Semantic diagram, the query language, the rule specification language, and the underlying formalism of the system (i.e., association algebra). Chapter 3 presents some query optimization techniques, with special emphasis on conventional and semantic query optimization. Chapter 4 introduces knowledge rules and a knowledge rule classification scheme beneficial to semantic query optimization. It identifies the type of rules that is most beneficial to semantic query optimization. When covering integrity constraint rules, this chapter describes the association types of

OSAM* and defines the constraints within each type. Chapter 5 defines transformation rules to perform semantic query optimization and exemplifies the use of the transformation rules through a series of case studies. Finally, Chapter 6 briefly discusses cost estimation for semantic query optimization. Cost estimation is introduced in this chapter as a guide to future research. A full treatment of this problem is out of scope of this research. A discussion on future work and a conclusion are also presented in this chapter.

CHAPTER 2
THE OSAM* KNOWLEDGE BASE MANAGEMENT SYSTEM

This chapter gives an overview of the Object-oriented Semantic Association Model OSAM*, its query and rule specification languages, and the underlying algebra called Association Algebra. The conceptual understanding of the model and the languages are necessary for the understanding of query optimization issues and techniques to be presented in the subsequent chapters.

Overview of OSAM*

The Object-oriented Semantic Association Model (OSAM*) is a semantic data model which is rich in constructs for explicitly specifying the structural properties of objects found in an application. In addition, it allows user-defined operations and knowledge rules to be specified as part of the semantic properties of objects. OSAM* is an integration of the object-oriented paradigm, database management technology, and rule-based system concept. From the object-oriented paradigm, it incorporates familiar objects (complex data types), the encapsulation of methods and types within an object, the inheritance of rules, attributes, operations, and associations, and the uniform treatment of data and meta-data. OSAM* also allows objects to inherit properties from multiple

parent classes and allows instantiation of an object in more than one class [SU86] and [SU89].

In OSAM*, objects sharing common semantic properties are grouped together to form an object class. Objects in a class have similar structural and behavioral properties. The definition of an object class consists of associations, operations, and rules. The associations identify the object class' structural relationships with other classes. The operations and their corresponding methods define the meaningful operations for manipulating a class' members. Finally, the deductive rules and integrity constraints specified for a class respectively generate data that is not explicitly stored in the knowledge base and govern manipulations performed against the class to ensure that they do not violate the semantic integrity and security of the knowledge base.

An object class is either an entity class (E-class) or a domain class (D-class). An E-class is used to model objects of an application that are independent entities (real world objects). Objects in an E-class are assigned object identifications (or OIDs). An E-class is materialized as instances of objects in that class. A D-class, however, serves as descriptive data of objects in E-classes or other D-classes. A D-class specifies domain characteristics such as data type, ranges of values, data structures, etc. The data values that form a D-class identify the objects in the class.

The semantic relationships among classes and thus their corresponding objects are defined in OSAM* by semantic associations and data type constructors. There are five system-defined association types: Aggregation (A), Generalization (G), Interaction (I), Composition (C), and Cross-product (X). These types are exemplified and explained in detail in Chapter 4. User-defined association types are also allowed. Data type constructors, on the other hand, are definitions of sets, vectors, matrices, etc. and form domains having complex structures such as set of real numbers, vector of integers, matrix of part numbers, etc.

The Semantic Diagram

The Semantic Diagram (S-diagram) is a network of nodes and edges that graphically represent the schema of a knowledge base in terms of classes and their associations. The S-diagram also presents frequently defined constraints associated with classes via specific labels and graphic symbols. The S-diagram does not show, however, operations and deductive rules that can be specified in a class. Most of the figures in this thesis are S-diagrams representing various example database schemas.

S-diagram primitives are as follows:

(1) Domain Class: A domain class (D-class) is represented by a circular node. The name of the D-class is presented next to the circle. In Figure 1, section#, room#, textbook, ss#, name, etc. are D-classes. Following the name can be a

system-defined data type or a constraint within a data type, such as a range or set of valid values. For example, in Figure 2 the D-class Name is typed as having character string of length 40. A D-class can also be defined in terms of other classes through an aggregation or a generalization association, or a data type constructor. Such classes are composite D-classes. Finally, attribute names that are different from their domain names are written as labels on the connecting edges of the D-class, such as the attribute names Major and Minor off the Student and Undergrad classes, respectively, in Figure 1.

(2) Entity Class: An entity class (E-class) is represented by a rectangular node in the S-diagram with its name appearing within the rectangle. In Figure 1, Section, Person, Transcript, Teacher, etc. are E-classes. Edges of E-class are labeled (at their originating endpoints) according to the association types the edges are representing (labeled with A, G, I, C, or X for the five system-defined association types). As with D-classes, attribute names that differ from the class names are shown via labeled edges.

(3) Constraints: There are many types of constraints that can be specified in an S-diagram. Different association types may have different types of constraints. For example, arcs across E-class edges represent a specified constraint on the associations those edges represent, such as a mapping constraint or a composite key constraint. A mapping constraint in Figure 1 is represented on the arc between Student and Course's links to the Transcript class. It specifies that a

student can take many courses and a course can have many students (i.e., a n:m mapping). Chapter 4 explains all association types and their constraints.

In summary, the S-diagram serves as a graphical tool to display the structural properties and frequent constraints specified in the schema of a knowledge base. An in-depth analysis of the S-diagram is given in a work by Su and Siggelkow [SU88a].

Object-oriented Query Language

OSAM*'s query language, an Object-oriented Query Language (OQL), is designed to produce results that are structurally and conceptually consistent with the knowledge base the query is executed against (i.e., having the closure property). The key feature of this language is that it is pattern-based language rather than the conventional attribute-based languages. Complex search conditions can be specified in this language it terms of patterns of object associations. The language allows complex queries to be specified in a relatively simple way. An OQL query results in a sub-database of some selected object classes and their associations. The sub-database is materialized by objects in the database that satisfy some association patterns given in the OQL query. The resultant sub-database of a query forms a context in which the system-defined or user-defined operations specified in the query can be performed on the selected objects. An OQL query is syntactically structured to allow the user to first define the appropriate sub-database applicable to the query and then to specify

the operations to be performed within that sub-database. Syntactically, an OQL query consists of two clauses: a CONTEXT clause and an Operation clause, as shown below:

CONTEXT association-pattern [intra-class conditions]
WHERE inter-class conditions
SELECT object classes and/or attributes
Operation(s) object classes

The CONTEXT clause specifies the sub-database the query is to be processed against. The association-pattern is specified in terms of a structure of object classes, association and non-association operators, and AND/OR branch operators. Each class in the structure is optionally followed by intra-class conditions expressed in the form of predicates. The association operator "*" specifies that objects in two adjacent classes that are associated with each other shall be retained in the sub-database. The association operator "!" specifies that objects in two adjacent classes that are not associated with each other shall be retained in the resultant sub-database. The association-pattern of the CONTEXT clause may also contain branches with AND and OR operators. These operators must procede a parenthesized list of sub-expressions, where the sub-expressions are an association operator and a class name, or simply a class name (in which case the applicable association operator, which applies to all the classes in the list, must proceed tne AND/OR operator). The AND/OR operators specify the AND/OR condition among objects of the involved classes. For example, the pattern (Section * AND(Teacher, Course)) specifies that a

section must be associated with both a teacher and a course. All section, teacher, and course objects that satisfy the AND pattern are retained to form the resultant sub-database.

The WHERE sub-clause of the CONTEXT clause is optional and can be used to specify inter-class comparison conditions. The comparisons must be between type-compatible attributes or between type-compatible objects of two classes. Object comparisons are limited to "equal" (=) and "not equal" (!=). Finally, the SELECT sub-clause specifies the portion of the resultant sub-database that is to be "projected" into the final resultant sub-database. Object classes and/or attributes specified in the SELECT sub-clause are retained in the resultant sub-database.

The Operation clause of the OQL query syntactical structure defines the operations to be performed against the final resultant sub-database of the CONTEXT clause. The operations can be system or user-defined and are activated via messages to the classes of the sub-database. Example of system-defined operations are DISPLAY and PRINT, for displaying and printing the values of the descriptive attributes appearing in the resultant sub-database, respectively.

OQL also provides some advanced features such as identification of sub-expressions within the association pattern, set operations on association patterns, and queries with multiple association patterns. As this thesis does not use any

of the advanced features, they will not be described. Refer to [ALA89a] for a complete discussion of an OQL.

In summary, OQL defines queries in two conceptual components: (1) a component to establish a sub-database context and (2) a component to specify the operations to be performed against that sub-database. The two-component structure is reflected in the two-clause format of an OQL query. As a patten-based language, OQL allows complex queries to be specified in a relatively simple manner.

Rule Specification Language

OSAM*'s rule specification language is a powerful, association-based language designed to derive data not explicitly stored in the knowledge base (deductive rules) and to ensure the knowledge base remains in a secure and semantically consistent state (integrity constraint rules). Though Chapter 4 categorizes and exemplifies the knowledge rules, this section introduces the constructs of the rule specification language, specifically the constructs pertinent to this thesis. The syntactical structure of a knowledge rule is:

```
RULE rule-id
TRIGGER_COND trigger-specification
rule-body
CORRECTIVE_ACTION action-specification
END
```

The full rule construct contains details that are superfluous to semantic query optimization, such as the rule-id, trigger condition, and corrective action.

The rule-body contains the knowledge that is pertinent to optimization. The body can have one of two formats: IF-THEN-ELSE or a simple statement of conditions that must exist in the knowledge base. The IF-THEN-ELSE format follows the typical structure:

IF antecedent
THEN consequent1
ELSE consequent2.

The antecedent of the rule body is an association pattern, just as in an OQL query's CONTEXT statement. It stipulates the conditions in which consequent1 of the rule must be maintained. For those portions of the knowledge base where the conditions of the rule's antecedent are found to exist, consequent1's conditions/operations must be upheld/performed. Instead of specifying the conditions that must exist or the operations to be performed, the consequent may define a sub-database or attribute(s) that is derived from the existence of the antecedent's conditions (i.e., in the case of deductive rules). Consequent2 in the ELSE clause is applicable in cases where the conditions of the antecedent do not exist.

The second type of format for the rule body is a statement of conditions that must be unconditionally upheld. Rule bodies of this format type either stipulate an association pattern that must or must not exist (through the EXIST and NOT_EXIST operators, respectively) or stipulate a particular attribute predicate that must be maintained.

In summary, the rule specification language of OSAM* defines knowledge rules for a given knowledge base. The rule bodies, like OQL queries, are association-based rather than the conventional attribute-based rules. Thus, the language allows comparably easier definitions of complex constraints and data derivations.

Association Algebra

Association algebra (A-algebra) serves as a formal foundation for the establishment of an OQL and rule specification language. Its association-based operators match and manipulate association patterns among data objects as to produce other association patterns. Association patterns specify conditions to select some objects for query or rule processing. Association pattern operators include: ASelect (σ), AProject (π), Associate (*), AComplement (|), AUnion (+), ADifference (-), ADivide (÷), NonAssociate (!), and AIntersect (•). Of these nine operators, this thesis explicitly employs Associate (*), NonAssociate (!), AUnion (+), and AIntersect (•) in its discussions. They will be described in detail. All association operators are discussed in a report by Guo, Alashqur, Lam, and SU [GUO89].

Before explaining each operator, some preliminaries must be covered. Assume that all objects are associated with themselves by what is called an

inner-association-pattern (Inn-pattern).[1] An explicit relationship between two Inn-patterns (i.e., two objects) is termed an inter-association-pattern (I-pattern). Objects not explicitly related to each other (i.e., not connected by an I-pattern) are conceptually associated by a complement edge. A complement edge and the objects connected by it form a complement-association-pattern (C-pattern). Derived-association-patterns (D-patterns) represent patterns of object that may be directly or indirectly associated, thus consisting of both complement and regular edges between objects. Associations between objects form an object graph, whose connected sub-graphs are association patterns. An Association-pattern-set (association-set) is a set of patterns or an empty set that does not contain any duplicate patterns. The operators of A-algebra are defined according to the possible relationships between the patterns of association-sets. Figure 3 is an object diagram that serves as an example in the definition of the following operators:

(1) Associate (*): The Associate operator identifies the association of objects of two classes. From the two association-set operands, application of the Associate operator generates a new association-set of those Inn-patterns and I-patterns where the I-patterns connect the Inn-patterns of the two operands' association-sets. For example, in the object diagram of Figure 3 (A * B) results in the association-set: (a1b1), (a2b4), (a3b2), (a4b5).

---

[1]This assumption allows the algebra to operate against single objects as well as objects' associations with other objects.

(2) NonAssociate (!): As a complement to the Associate operator, the NonAssociate operator identifies a set of objects of either operand association-set that do not connect to any association-patterns of the other operand association-set. (A ! B) for example results in the a5 and b3 objects in the object diagram example.

(3) AUnion (+): The AUnion operator combines two association-sets to produce all associations of both sets. Redundant and superceded association-patterns are eliminated. An association-pattern is superceded by another if it is contained within the another association-pattern. The pattern (A * B + B * D) results in the association-set of (a1b1), (a2b4), (a3b2), (a4b5), (b4d1), (b5d2) in the object diagram.

(4) AIntersect (•): The AIntersect operator is similar to the conventional intersection operator of logic but is applicable in the context of association-sets. Its resultant association-set contains a set of associations which have at least one association from both operand association-sets and where these two associations share a common inner-association in each of the common classes of the operand association-sets. The resulting association-set of (A * B * C • A * B * D) is shown in Figure 3. The b4 object represents the common inner-association of the (A * B * C) and (A * B * D) operands. The resulting association-set is (a2b4), (b4c2), (b4d1).

The logical properties within association algebra of commutatively, associativity, idempotency, identity, and distributive properties [GUO89] are not explicitly employed in this thesis but will be important in the implementation of semantic query optimization in OSAM*.

In summary, association algebra provides the formal basis to establish an OQL and a rule specification language. It permits logical operations on associations between objects. The logical operations are important in the association patterns specified in an OQL query CONTEXT clause and in a constraint rule's antecedent and consequent.

In conclusion, this chapter has reviewed an underlying KBMS for the OSAM*. The data model of the system, the graphical tool for schema specification, the query and rule specification languages for the system, and the underlying formalism of these languages have been discussed. The purpose of this discussion was to establish a context in which semantic query optimization is to partake.

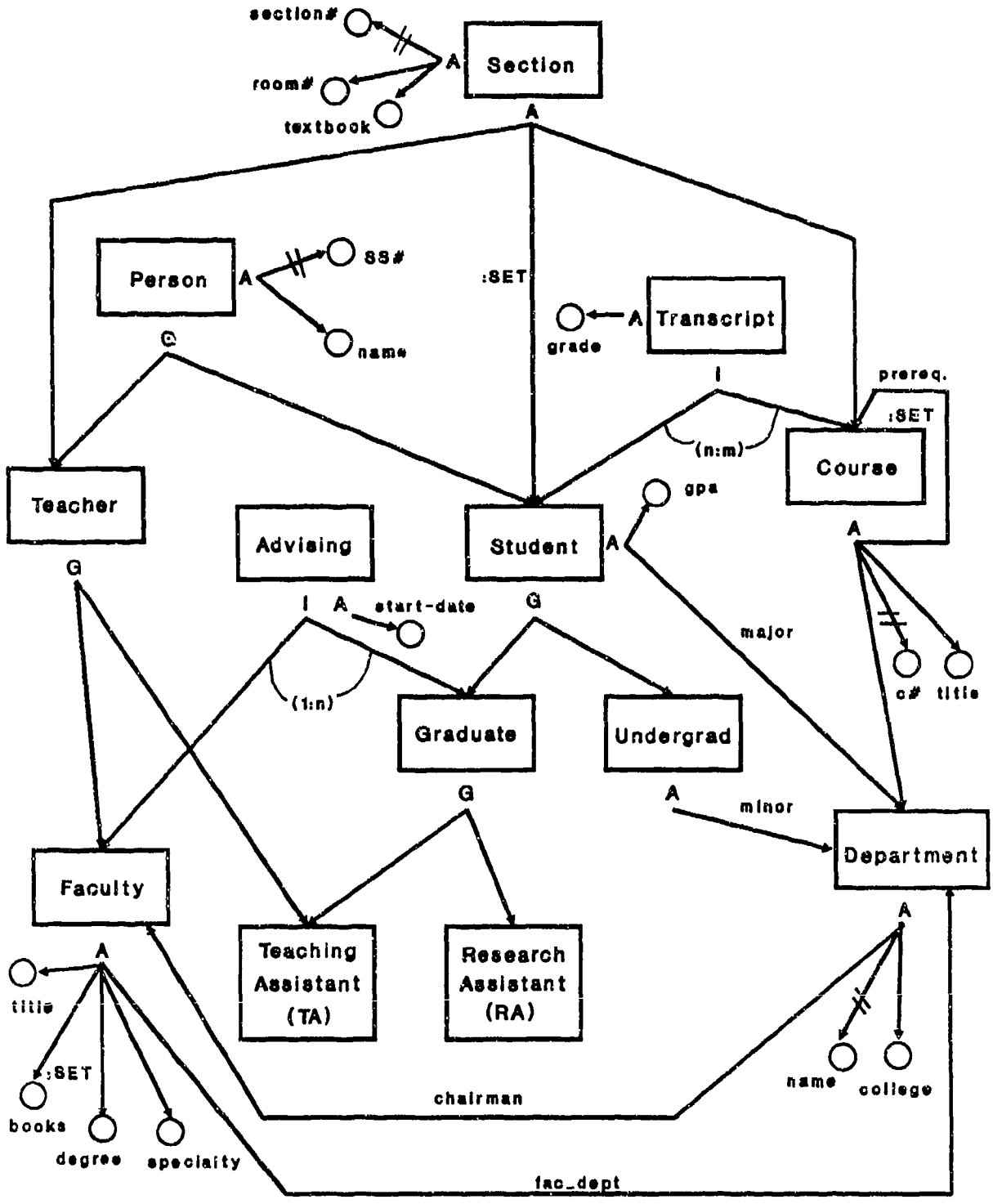

Figure 1. University Database Schema.

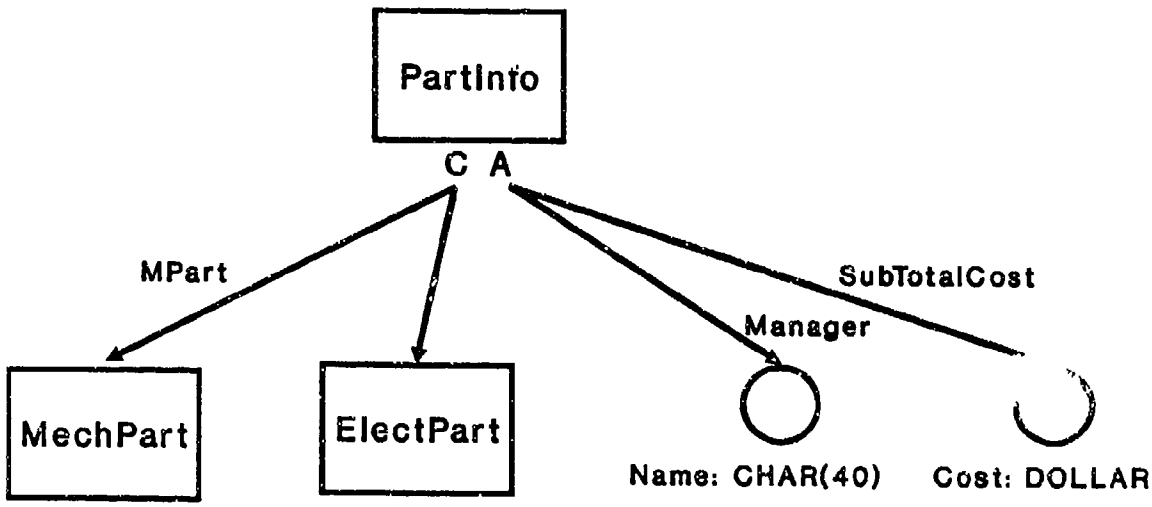

Figure 2. Parts Database Schema.

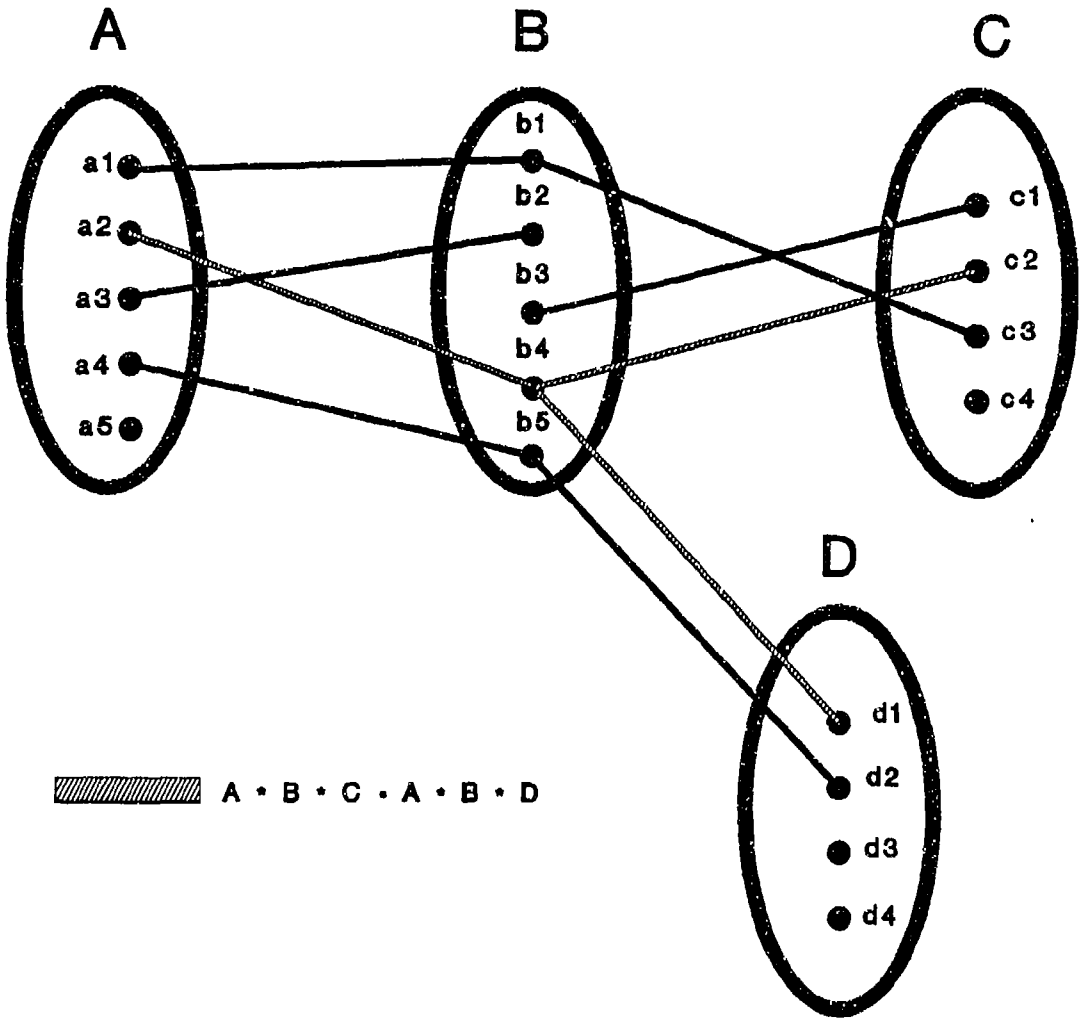

Figure 3. Object Diagram of Classes A, B, C, D.

CHAPTER 3
QUERY OPTIMIZATION

Query processing is a multi-stepped procedure. Upon receiving a user-submitted query, the Database Management System (DBMS) or Knowledge Base Management System (KBMS) should validate, translate, optimize, and then execute a query. Query validation ensures syntactic and semantic correctness of the query, resolves what portion of the database the query references, and performs security checks to ensure the user is permitted the requested operations. Once validated, a query can be translated into an internal representation understandable to the DBMS/KBMS. The DBMS/KBMS can then apply optimization techniques to generate an efficient processing strategy. Finally, the query is executed to obtain the desired results. It is the third step of this multi-phased process, i.e. query optimization, that shall be examined in this chapter.

Overview of Query Optimization Techniques

The introduction of non-procedural query languages have made queries easier to formulate and more user-friendly. In doing so, the languages have alleviated the user from efficiency concerns, such as the selection of the access

method and the order to perform the primitive operations. The user can concentrate on modeling the real world application and state what he/she wants from the database rather than describing how data should be obtained. This concept is called data independence; meaning, the end user is free from the internal and physical data storage details and can design the system that is conceptually appropriate for the application. Codd's introduction of the relational data model allowed data independence to be fully realized [COD70]. While data independence and non-procedural query languages make a system more user-friendly, it places the burden of achieving efficiency within the internal processing of the DBMS. The distinction between the logical and physical representations of the database has forced what was once the responsibility of the application programmer to become the responsibility of the DBMS/KBMS.

Query optimization is a broad field of study within DBMS/KBMS systems that examines methods to make query execution as efficient as possible. The goal of query optimization is to translate a user-submitted query into an efficient query evaluation plan (QEP). A QEP is an executable representation of how a query is to be efficiently processed as to return the requested results. Frequently, there are many ways to process a query to produce the correct results. The query optimizer would generate multiple QEPs from which the most efficient plan is selected for execution. This methodology brings up two important points.

First, the KBMS/DBMS must have a selection criteria to choose from the multiple QEPs generated. The selection criteria is based on costs incurred in processing a query. In general, the cost may include secondary storage access cost, computational cost, cost of physical storage (main memory and secondary storage), and communication cost. The costs are not independent from each other, as they are often determined by some related factors. Specifically in the processing of a query, factors such as the amount of data being processed, the organization of the data, the order of operations, and knowledge about the data itself can greatly effect the above costs. Query optimization, then, must examine such factors and streamline their effect. However, since query optimization is in itself additional processing, the cost of the optimization techniques must be significantly less than the saving that will be realized by performing the optimization. In short, a cost-benefit analysis of optimization methodologies must be conducted. Such an analysis is outside the scope of this thesis.

The second point of query optimization generating multiple QEPs is that "optimality" is most often an unreachable goal. The reason for this shortcoming is two-fold. Firstly, if every possible QEP is generated for a query, then the cost of optimization can be much greater than the savings incurred in doing the optimization. It is computationally intractable to do such an exhaustive generation and search for optimality [FRE87]. The generation of QEPs must therefore be focused on those plans that offer the greatest potential for saving

query processing time and require only minimal computational time to generate[1]. Secondly, no existing system implements all aspects of optimization, nor is it possible to since some techniques are alternatives to each other. A few of the techniques include rule-based, dynamic, multiple query, conventional (or syntactic), and semantic optimization. These means are explained below:

(1) Rule Based Query Optimization [FRE87]. In rule based query optimization, rules define how to derive different QEPS and restrict the number of QEPs generated for a query. Many experimental/research systems employ heuristic rules to guide the translation process of a query into QEPs that have good optimization potential. Heuristics are necessary since finding the optimal QEP is intractable (as already discussed). Rules commonly used in optimization pertain to access methods, ordering of operations, predicate distribution, join strategies, and use of indexes.

(2) Dynamic Query Optimization [GOE89]. While most queries are optimized at compile time, dynamic query optimization occurs as a query is being executed. Optimization at compile time requires estimations on variable values, available resources, size of intermediate results, and the existence of access paths. Dynamic optimization, however, can use current parameters of these variables to decide on appropriate strategies. Dynamic optimization can also be used to

---

[1]Query optimization is a misnomer. It really means query amelioration rather that optimization, since a significant improvement, not optimality, is understood to be the underlying and achievable goal.

"learn" from past performance. As optimization procedures choose a strategy, statistics can be kept on how the choice fared; that is, how much savings is gained compared to the cost incurred to achieve those savings. The statistics can then be used in future optimization decisions.

(3) Multiple Query Optimization [CHA85]. Queries are often envisioned as single entity requests for information. However, it is quite likely that a series of queries run consecutively are inter-related and often perform the same operations and request for much the same information. Thus, the intermediate results of one query may be useful to a following query. Combining the evaluation of queries with common sub-expressions can be beneficial and time-saving. Multiple query optimization attempts to optimize groups of queries to avoid evaluating common sub-expressions more than once. In short, multiple query optimization aims to reduce duplicated effort.

Finally, conventional query optimization and semantic optimization are discussed in detail in the next sections, as they are two more promising and popular techniques. Keep in mind that no system implements all five techniques, and likewise, few systems employ only one aspect. Most systems employ a two or three faceted optimization technique, such as a combination of rule-based, conventional, and semantic optimization.

Conventional Query Optimization

Conventional query optimization, also called syntactical query optimization, logically transforms a query into multiple QEPs, relying on the operator properties and storage details to perform the transformations. Most often, relational calculus is the underlying mathematical formalism to logically generate the QEPs. First, the query is converted into first order predicate calculus, then QEPs are generated through algebraic transformations, and finally, the "cheapest" QEP is executed. The algebraic transformations used to generate the logically equivalent QEPs are guided by heuristic rules that consider existing access paths, data structures, and algorithmic efficiencies. The heuristic rules, or "rules of thumb," undertake these considerations to perform optimization.

In considering access paths and data structures, conventional optimization must compare the type of scan being performed to the available accesses and data structures, including such items as indexes, B-trees, hash tables, sorted data files, etc. Obviously if a full scan of a file is being performed, the alternative access paths listed offer little savings. However, data specific searches can take much advantage of the efficient access paths and special data structures. Good conventional optimization may track access frequencies and types of data scans as to determine if new access plans/data structures are warranted. It will then suggest to the database administrator that an alternative is needed or will automatically introduce the alternative in a self-adaptive manner.

Typically, there are several efficiencies considered by conventional query optimization. Most of the consideration is focused on the join operator since the join is recognized as a frequent and the most expensive operation. The multitude of join algorithms [SU88b], including nested-loop join, merge join, hash join, and semi-join, indicates the large amount of study devoted to this operation and its cost of computation. The selection of an efficient join algorithm not only depends on the amount of data being joined, the data structures, and the efficient access paths available, but also on the hardware configuration of the system on which the DBMS/KBMS is implemented [SU88b]. Conventional query optimization may also consider low level details, such as storage methods like data buffering and clustering, to introduce additional data access efficiency.

Finally, because of the complexity and intractability of query optimization, conventional optimization relies on heuristic rules to successfully limit the scope of its processing. The rules perform modifications based on operators of the query language. That is to say, the rules do not consider knowledge about data domains and relationships, but rather focus on heuristic guidelines derived from statistical data kept on query language operators and their probabilities. The rules pertain to the following strategies:

(1) Scanning strategy. Heuristic rules can consider access methods for particular data, choosing a strategy based on the amount of data being accessed, physical

representation of the data, and statistical information on the distribution of attribute values.

(2) Join strategy. Heuristic rules can choose join algorithms, access paths to joining data, order in which join is performed, and the outer and inner data groups of the join. Rules may also indicate when a join is not necessary and can be eliminated.

(3) Ordering of operations and early application of data reduction operations. Heuristic rules determine when to apply restriction and projection operations so as to move them as early as possible in processing. Early application of these operations will reduce intermediate data size. Rules can also determine if distributing predicates among all involved data is possible, again to reduce the working data size. Pushing down predicates prior to join operations in order to reduce the amount of joined data is also a rule that is frequently applied. Finally, rules may also detect null answer query selections.

In summary, conventional query optimization examines access paths, data structures, and algorithm efficiencies to perform algebraic transformations of a user-submitted query into a multitude of QEPs with the guidance of heuristic rules. It is therefore characterized by its reliance on syntactical information and implementation details to generate QEPs.

Semantic Query Optimization

Semantic query optimization is different from conventional in that it uses semantic knowledge about data to generate QEPs and improve query execution. Semantic query optimization is the process of applying a set of semantic query transformations that result in a set of semantically equivalent queries and choosing the one with lowest execution cost [SIE88]. Semantically equivalent queries produce the same results when applied against a valid database state, even though the queries may be markedly different. Semantic equivalence is weaker than the notion of logical equivalence. A set of logically equivalent queries are contained in the set of semantically equivalent queries for any query and database. Note that conventional query optimization is solely concerned with generating logically equivalent QEPs. Semantic query optimization, however, relies on data relationships, domains of data instances, and database state constraints to perform query transformations. Such information is readily available in a knowledge base.

A Knowledge Base Management System (KBMS) is an extension of the traditional Database Management System (DBMS). It not only stores elementary facts (the actual data) but also manages the general laws pertaining to those elementary facts. The general laws in a KBMS govern operations against the elementary facts to ensure the variable types are not violated, that user-specified rules concerning the data and data relationships are upheld, and

that the requested operations are valid and permitted. The elementary facts of a KBMS form the so-called extensional database (EDB). In contrast, the meta-data and general laws of a KBMS are called the Intensional Database (IDB). Note that while the EDB is very dynamic, the IDB is fairly static. The IDB is the conceptual definition of the knowledge base. It changes only when the semantics of the application being modeled change. The knowledge in the IDB is used in conjunction with query processing and optimization. It is compiled into an internal format for use by a query optimizer prior to processing any query.

The IDB is a rich source of semantic knowledge that is useful for producing semantically equivalent queries. Specifically, the knowledge rules specified in the IDB that are defined to maintain a valid knowledge base state or to derive new information serve quite well to improve query processing. The knowledge within a rule can transform a query by the addition or deletion of constraints to the query or by the replacement of requested extensional data of query with derived data. The transformation set is the set of all rules that can be used to alter the query. Obviously, a knowledge base with many rules can lead to larger transformation sets, which in turn, cause higher semantic optimization cost. The generation of the transformation set must therefore be guided by heuristics to limit QEP generation to meaningful and profitable plans.

The KBMS must employ control strategies to avoid generating a large number of semantically equivalent queries in an exhaustive manner.

In order to use knowledge rules in query processing, a user-submitted query and the rules must have comparable formats; that is, the system must be able to integrate a rule's contents with the constraints specified in a query. Most KBMSs use the relational data model as their means for database definition. Outside of the structural specification of a database, the user must specify rules and operations for the defined database. The query language is specifically designed for the relational data model. The common underlying formalism for relational query languages and their knowledge rules is first order theory and predicate calculus. Through this formalism, the semantics of rules are integrated with queries to generate semantically equivalent queries.

Given a common underlying formalism, semantic query optimization becomes a two step process: semantic compilation and semantic query transformation [CHA85]. Semantic compilation is the process of converting a knowledge rule into an internal representation that is useful to the KBMS. For the relational-data-model-based KBMSs, the transformation set for any given query must first be identified and then the rules in the set must be converted into the internal representation. When a query is compiled or translated, it must also undergo this transformation. This internal representation implements first order theory and predicate calculus. Semantic query transformation, the

next step in semantic optimization, is the integration of integrity constraints and deductive rules with a query in order to generate a set of QEPs that are projected to be more efficient than the original user-submitted query. Again, heuristic procedures must guide this step of the optimization process to ensure that the transformations will be profitable (more efficient and not difficult or too costly to derive).

In OSAM*, the structure of the KBMS is different than in the relational-data-modeled KBMSs. The EDB is the materialization of objects in object classes and their associations with other objects. The IDB consists of the structural relationships among object classes, the methods (or operations), and knowledge rules (deductive or constraint rules) associated with the classes. Integrity constraint rules can be implicitly and structurally represented by key-words in the S-diagram or explicitly defined by a knowledge rule definition language. In order to integrate knowledge rules and a query in OSAM*, the rules and user-submitted query must have comparable contexts; that is, the context of the rules must be akin to the portion of the data base referenced in the query. Unlike the relational-data-model-based KBMSs, the transformation set in OSAM* needs not be sought. Knowledge rules are defined explicitly within their object classes and therefore directly associated to the object classes being reference in a query. The rules simply need to be compiled into their internal formats.

As outlined in chapter two, OSAM* is a much more powerful modeling tool than the relational data model. It not only defines the structural properties of object classes which include several system-defined association types, but also captures the behavioral properties in terms of operations and knowledge rules. Its query language (OQL) therefore focuses on patterns of object associations. The underlying formalism for OQL is the association algebra [GUO89]. Association algebra serves well as a common underlying formalism for representing queries and knowledge rules for a KBMS based on OSAM*, just as first order theory and predicate calculus does for the relational-data-modeled KBMSs.

The semantic query transformation step aims to alter a query in lieu of semantic information. Frequently, alterations are either semantic expansion, semantic reduction, or semantic replacement [SHE87]. Semantic expansion is the addition of a new restriction implied by the combination of a query and integrity constraint. Specifically, a rule whose antecedent is satisfied by constraints specified in the query can incorporate its consequent as another restriction to the query, provided this transformation is deemed profitable and the restriction is not already present in the query. As semantic query transformations are being performed, it is advisable to attempt early detection of contradictions in restrictions, as they immediately imply a null answer to a query. Semantic reduction aims to detect redundant restrictions. Specifically, if the

constraints specified in a query imply the antecedent of a rule and the rule's consequent is implied by a restriction already present in the query, then the restriction is redundant and may be removed from the query, provided that the removal does not leave an empty context and that it is profitable. Semantic replacement is the exchange of derived data from a deductive rule for extensional data requested in the query. The amount of derived data must be less than the extensional data and must be materialized prior to query processing. Specific examples of semantic query optimization in OSAM* are detailed in Chapter 5.

In summary, the goal of semantic query optimization is the generation of semantically equivalent and more efficient QEPs. The integrity constraints and deductive rules of an intensional database encapsulate knowledge useful for this transformation purpose. In order to use such knowledge, however, the queries and rules must have a common underlying formal basis that will serve to integrate the two. Semantic compilation converts rules into an internal representation. The rules and a query are then integrated in semantic query transformation step to generate QEPs projected to be more efficient than the original query.

As a final note on query optimization, very seldom is a single optimization technique employed, as already mentioned. An example of an integration of techniques is the combination of rule based, semantic, and

conventional query optimization aspects. Thus, query optimization would consist of first searching for semantically equivalent queries (semantic compilation and query transformation), guided by a set of heuristic rules representing query processing expertise and costing information. Each semantically equivalent candidate query would then employ conventional optimization techniques (syntactic transformations) to further introduce lower level QEPs. As such, the number of candidate QEPs can grow exponentially if a good search control is not implemented in the KBMS. Each of the techniques must interact with the others in order to guide the whole process towards profitable transformations. The importance of search control cannot be overlooked when devising an overall optimization strategy.

In conclusion, the separation of the physical database from the conceptual database has brought forth ease in designing and using database systems. However, it has shifted the responsibility of efficient query processing to the DBMS/KBMS. To uphold this responsibility, the DBMS/KBMS may employ rule-based, dynamic, multiple-query, conventional, and semantic query optimization techniques. Conventional and semantic query optimization have been discussed in detail since they represent the most promising and popular aspects of query optimization.

CHAPTER 4
RULE TYPES AND EXAMPLES

There are two main types of rules in a Knowledge Base Management System (KBMS): deductive rules and integrity constraint rules. The purpose of the following sections is to define each rule type, to give examples of such rules, and to explain each type's potential to help optimize queries. Within the next sections, each type is further partitioned according to a rule's functionality within their type.

It should be noted that none of the rule examples given specify a trigger condition or a corrective action. Their exclusion is intentional, since the rule body (IF-THEN portion of the rule) contains the knowledge useful for semantic query optimization.

Deductive Rules

A deductive rule is defined as follows: A deductive rule is a knowledge rule whose application against a knowledge base results in the generation of information that is not explicitly stored in the knowledge base. Integrity constraint rules, on the other hand, ensure the semantic correctness and consistency of the data that is explicitly stored in the knowledge base. Integrity

constraint rules can ensure semantic correctness by either representing valid states in the knowledge base or by performing operations to maintain valid states. Constraint rules are discussed in the next section. In OSAM*, a deductive rule can be further categorized by what the rule deduces. The rule may deduce a sub-database, attributes, or a combination of sub-database and attributes. The following paragraphs refine these categories. The university model in Figure 1 is the S-diagram used for deductive rule examples.

Deduction of Sub-database

Rules of this category derive a sub-database through implied relationships between classes or through specific restriction on an existing class(es). These rules specify the name and attributes of the resultant sub-database in their consequent. A rule substantiating an implied relationship derives associations not explicitly modelled in the S-diagram and projects the desired attributes for the resultant sub-database. Rule 1 is a deductive rule of this type.

Rule 1:
IF Teacher * Section * Course
THEN Teacher-Course(Teacher[ss#,name],Course[c#,title]).

Explanation: Because a teacher teaches a section of a particular course, the teacher is indirectly associated with the course. Supposing we want to know what courses a teacher instructs but are not interested in the sections of such courses, application of the above rule creates a sub-database containing teacher

instances which are directly associated with course instances. The class Sections will not be contained in the resultant sub-database.

A deductive rule can also generate a sub-database via a restriction on the instances of an existing class(es). Rule 2 is a rule of this type:

Rule 2:
IF Student[gpa > 3.5]
THEN Honor-Student(Student).

Explanation: The above rule does not deduce any associations but rather creates a sub-database called Honor-Student based on the restriction on an already defined class (Student).

Deduction of Attributes

Rules in this category represent the derivation of attribute values that are not explicitly stored in the knowledge base. The attribute values may be dependent on other attribute instances, as in Rule 3, or may be the computational result of operations on instances in the knowledge base, as in Rule 4:

Rule 3:
IF Faculty[degree == "PhD"]
THEN title := professor ELSE title := "teacher".

Rule 4:
IF Student * Transcript * Course
THEN gpa := SUM (Grade by Student)
DIV COUNT (Course by Student).

Explanation: For Rule 3, it is assumed that there is a derived attribute called "title" in the Faculty class. The attribute is directly dependent on a faculty member's degree, where such a member should be titled "professor" if he/she has a PhD, otherwise titled "teacher." Rule 4 illustrates how gpa could be a derived attribute that is calculated from a student's grades, assuming grades are stored numerically instead of alphabetically. The rule employs SUM, DIV, and COUNT operations to accumulate a student's grades and then divide them by the number of courses he/she has taken.

Combination of Sub-database and Attributes

Finally, a deductive rule can derive a combination of sub-database and attributes.

```
Rule 5:
IF Faculty[degree == "PhD"] * Section * Course
THEN PhD-Faculty-Course (Faculty[ss#,name,degree,
        title := "professor"], Course[c#,title]).
```

Explanation: The above rule combines the derivation of associations and new attributes such that the resultant sub-database contains faculty members with PhDs and the courses they instruct. Note that the members are appropriately titled as professors and that the new sub-database no longer references Faculty's relationship with sections to get to the courses they teach.

Deductive Rules and Query Optimization

In OSAM*, deductive rules are not explicitly represented in an S-diagram but rather are user-specified using a Knowledge Base Definition Language

(KBDL) and stored in the data dictionary with their appropriate classes. Additionally, a rule is defined to be data dependent if satisfaction of its antecedent requires the value of an object (i.e., the descriptive data or the associations of an object) to be accessed. For example, Rules 1 through 5 are data dependent. However, if Rule 2's antecedent did not contain the attribute restriction (gpa > 3.5), then it would be data independent. Given this definition, deductive rules will usually be data dependent. The concern of data dependence is one of efficiency to be considered at implementation. From a conceptual viewpoint, deductive rules are generators of new information. The sub-databases they generate must be fully operable databases in the sense that they can be queried, further deduced by other deductive rules, and constrained by integrity constraint rules.

Deductive rules are important to query optimization when a query or an integrity constraint rule references the derived data. The knowledge of how the data is derived, which is contained in the rule, may serve to optimize a query directly or may work in conjunction with an integrity constraint. For example, if a query references some derived data, yet the deductive rule that derives the data indicates that the query is not satisfiable, then obviously the operations to derive the data need not be performed. A null answer can be immediately returned. Rule 5 could be such a case. Supposing a query requests a list of professors that do not have their PhDs. The PhD-Faculty-Course sub-database

does not need to be derived, since the rule ensures that there would never be an instance in the knowledge base where a teacher titled professor does not have a PhD. A null answer could be quickly returned. An example of where an integrity constraint rule and a deductive rule may work in conjunction is as follows: If an integrity constraint rule references (in its antecedent or consequent) a sub-database or attribute generated by a deductive rule, that deductive rule must be considered in query optimization. The following integrity rule references the resultant sub-database of Rule 5:

Rule 6:
IF PhD-Faculty-Course
THEN NOT_EXIST (PhD-Faculty-Course * Teaching-Assistant).

Explanation: This integrity rule implies that a teacher with a PhD (alias, professor) is never a teaching assistant (TA). As such, the deductive rule must also be considered in query optimization as it contains knowledge of how the data is derived. Specifically, if a query requests for professors who are TAs, the integrity constraint rule combined with the knowledge in the deductive rule (Rule 5) can be used to immediately answer the query with a null response, without having to access the extensional database.

Finally, the derived data specified in a deductive rule can also be used to optimize a query. Specifically, the derived data may replace extensional data requested in a query. The derived data must be of smaller size and must be materialized prior to the processing of the query in order for this replacement to

be more efficient than processing the original query (i.e., accessing and extracting the requested extensional data). This transformation (i.e., Semantic Replacement) is demonstrated in Chapter 5.

Thus, query optimization with deductive rules may serve to forgo the operations necessary derive information or may serve to replace requested extensional data. When the operations to derive data can be eliminated, the savings may be significant if the amount of data being derived is large or if the operations to derive the data are complex. Likewise, when derived data can replace a request for extensional data, the savings may be significant if the amount of derived data is significantly less than the amount of requested extensional data and is materialized prior to query processing. The amount of savings versus the cost of performing the optimization step must be considered when generating optimization plans. This topic is discussed in Chapter 6.

As a final note on deductive rules and query optimization, an efficiency measure that can be implemented is the a priori generation of the sub-databases and attribute values derived from data independent deductive rules. When a query or another rule references the resultant sub-database or attributes, the information will have already been derived and available. Data dependent deductive rules cannot implement this efficiency measure because of their need for object instance values. Given this efficiency step, only data-dependent

deductive rules need to be considered in run-time query optimization; data independent rules can be compiled prior to query processing.

Integrity Constraint Rules

An integrity constraint (IC) rule is a knowledge rule designed to ensure semantic correctness and consistency of a knowledge base. Integrity constraints are applied when an operation against the knowledge base (i.e., a trigger condition) can potentially violate the knowledge base's semantic correctness and consistency. In OSAM*, integrity rules are specified in a schema in two ways. Frequently used integrity rules are specified by key words in the structural portion of an object class definition (e.g., an attribute is a key or the mapping between two classes is many-to-many). This method of rule specification simplifies the user's task since a set of implicit rules can be named by key words and can be structurally shown in an S-diagram. Other constraint rules can be explicitly defined in the constraint specification language by the rule portion of an object class definition.

IC Rules Represented in the OSAM* S-Diagram

The following paragraphs itemize the possible integrity constraints that can be specified in an S-diagram. These constraints can be mapped directly into integrity constraint rules. Examples of such mapping and the S-diagrams used in the examples are given. This itemization serves two functions. First, it enumerates the association types of OSAM* and the constraints within each

type. Second, it serves to familiarize the reader with the S-diagram and how the semantics of the diagram can be represented by integrity constraint rules. Within each association type, a definition for the type is given and the constraints within the type are listed. Following each constraint within a type, a rule example, the pertinent S-diagram figure number, and the meaning of the rule are given if such a rule can be constructed for that type of constraint.

Generalization

The generalization association type is used to indicate a super-class/sub-class relationship between two object classes, much like that of the object-oriented paradigm with a super-set/subset constraint between the super-class and sub-class. This relationship (i.e., "is-a" relationship) indicates an automatic inheritance of attributes, operations, and rules of the super-class to the constituent sub-classes. Figures 4 and 5 contain generalization association types. For example in Figure 4, FemaleEmp, MaleEmp, Engineer, and SupportEmp are sub-classes of the Employee super-class. Constraints within the generalization association type are:

(1) Super-class/Sub-class Constraint in Figure 4.

Rule 7:
IF Engineer THEN Employee * Engineer.

Explanation: The Engineer class is a sub-class of the Employee super-class. As such, an engineer must also be an employee (i.e., there must be an association between every engineer and employee).

(2) Set eXclusion (SX) in Figure 4.

Rule 8:
IF Employee * Engineer THEN Employee ! SupportEmp.

Rule 9:
IF Employee * SupportEmp THEN Employee ! Engineer.

Explanation: Either of these two rules will suffice to indicate that an employee cannot be both an Engineer and a SupportEmp. Specifically, the first rule finds employees that are engineers and then checks to ensure that they are not also supporting employees. The second rule is simply the reversal of Engineer and SupportEmp classes. Note that this constraint could also be specified as follows:

Rule 10:
NOT_EXIST (Employee * AND(Engineer, SupportEmp)).

Explanation: This rule explicitly states that an employee must not be specified as both an engineer and supporting employee.

(2) Set Intersection (SI -- default).

(3) Set Equality (SE): different perspective of the same object, Figure 5.

Rule 11:
IF Part * Geometric THEN Part * Machining.

Rule 12:
IF Part * Machining THEN Part * Geometric.

Explanation: Every part that is associated with a geometric part must also be associated with a machining part and vice versa. Note that both Rules 11 an 12 are necessary to ensure that the set equality constraint holds. Neither by

itself fully guarantees semantic correctness. It will become apparent in Chapter 5, Case Study 4 that single semantic properties should be written as a single rule to be more beneficial to semantic query optimization. As such, rules 11 and 12 can be more correctly written as follows:

Rule 13:
IF (Part * OR(Geometric, Machining))
THEN (Part * AND(Geometric, Machining)).

Explanation: This rule is closer to the exact semantic meaning of set equality. It implies that if a part has either geometric information or machining information, then it must have both types of information.

(4) Total Specialization (TS): super-class is the union of subclass instances as shown in Figure 5.

Rule 14:
IF Container THEN Container * OR(Device, Tray).

Explanation: Every Container must be associated with a Device or Tray. Again, this rule may be written in a non-IF/THEN format:

Rule 15:
NOT_EXIST (Container ! OR(Device, Tray)).

(5) Partial Specialization (default): some instance of a super-class does not belong to the subclass(es).

(6) Specialization Condition as shown in Figure 4.

Rule 16:
IF Employee[Sex == "F"] THEN Employee * FemaleEmp.

Rule 17:
IF Employee[Sex == "M"] THEN Employee * MaleEmp.

Explanation: An Employee specified as having sex equal "F" is a FemaleEmp; sex equal "M" is a MaleEmp.

Aggregation

Aggregation association type models the semantics of "has-a" relationship between two classes. The class being defined has an attribute whose value is drawn from a constituent class, which may be either a domain class or an entity class. The objects of the constituent class form the domain of the attribute. In general, an entity class has a number of attributes defined over a number of constituent classes. One, or a combination of attributes, can uniquely identify the objects of the defined class. This unique identity serves as the "user-defined" key. This key is separate and distinct from the system-defined and maintained object identification (OID). OIDs are transparent and inaccessible to the user. For example, in Figure 6, the attribute X_Id defined over domain class X2 is pictorially identified as the user-defined key by the two cross bars on its association to class X2. Likewise, attributes Y1_attr and Y2_attr defined over classes Y1 and Y2 respectively, form a composite key for ClassY. This key is pictorially indicated by the two cross-barred arch spanning the attributes' associations.

(1) User-defined key in Figure 6.

Rule 18:
MAPPING (ClassX, X2, 1:1).

Explanation: Assume "MAPPING" is a function that takes a class, a class name, and a mapping specification (one-to-one in this case) as input and ensures that the instances within the specified classes meet the mapping constraint. This function can be used to enforce the fact that X_Id attribute is a user-defined key of ClassX. User-defined keys, however, are a special system function that is handled internally by the KBMS. Therefore the KBMS ensures that the key attribute values are unique. The rule for this type of constraint does not have to be explicitly defined by the user.

(2) Composite key in Figure 6. See user-defined key above.

(3) Total Participation (TP): every object in a constituent class must be a participant in the defined class. Assume the class Section totally participates in the class Course in Figure 1.

Rule 19:
IF Section THEN Section * Course.

Explanation: All Sections must be assigned to (associated with) a Course.

(4) Non-Null Constraint (default) in Figure 6.

Rule 20:
z3_Attr != NULL.

Explanation: The value of the z3_Attr attribute must exist; it cannot have a null value. NULL is a system-defined value.

(5) Mapping (default is n:1): parent-to-child mapping. 1:1 mapping implies a candidate key. Same construct as the example for user-defined key.

Interaction

Interaction association type models the semantics of stating facts or relationships that exist between/among objects of two or more participating classes. Each instance of an interaction association type is formed by the objects of the participating classes. Interaction instances can take on attributes that represent descriptive properties. Figure 7 is used as an example of this association type. The schema in this figure indicates that suppliers interact with parts (i.e., they supply parts) and this interaction is described by a quantity and a date. Constraints within the interaction association type are:

(1) Total Participation (TP): every object in a constituent class must participate in some interaction as specified in Figure 7.

Rule 21:
IF Supplier THEN Supplier * Supply.

Explanation: All suppliers must participate in supplying parts.

(2) Mapping as specified in Figure 7.

Rule 22:
IF Supplier * Supply * Part
THEN MAPPING (Supplier, Part, n : m).

Explanation: Using the same MAPPING function as discussed in the user-defined key example in the aggregation association section, this rule states that one suppliers can supply many parts and one part can be supplied by many suppliers.

(3) Non-null: values that form an instance of an interaction cannot be null.

Rule 23:
IF Supply THEN Supply * AND(Supplier, Part).

Explanation: Every case of supply must have an associate supplier and an associated part being supplied.

(4) Defined over entity classes only.

(5) No inheritance.

Composition.

Composition association type is a construct for forming an object class that uses the constituent classes as its members. That is, the set of objects of a constituent class is treated as a single object in the composed class. Aggregation associations then apply to the resulting composition to define the descriptive data of the objects in the composed class. For example in Figure 2, the PartInfo class is a composition of MechPart and ElectPart; that is, the dynamic set of objects in MechPart is treated as an object of PartInfo and the dynamic set of objects in ElectPart is the other object of PartInfo. Manager and SubTotalCost are so-called "class attributes." Constraints within the composition association type are:

(1) Distinct object classes: constituent classes have to be distinct and contain dynamic sets of objects.

(2) Constituent classes must be E-classes.

(3) No inheritance.

Cross-product

Cross-product association type can be interpreted as the "grouped by" semantics. That is, the cross-product of objects in the constituent classes form the names of categories of objects. These categories can have summary attributes. Figure 8 is an example on a cross-product association type. The objects of Population_Group class are named by a cross-product of State, County, Age, and Sex objects. Each object in Population_Group class represents a category of objects which have summary attributes Count and AvgSalary. This association type is a special case of an aggregation association type. It is useful for modeling statistical databases. Constraints within the cross-product association type are:

(1) Constituent classes must be domain classes.

(2) All attributes form a composite key (uniqueness implied); same as user-defined key example of the aggregation association type.

(3) Non-null constraint. None of the attributes can have null values, which is the same construct as non-null constraint of the aggregation association type.

(4) Total Participation (TP -- default); same construct as with aggregation and interaction types.

Further Constraints

Below are general constraints relating to an S-diagram as a whole:

(1) Interaction, composition, and cross-product association types are mutually exclusive within a given class.

(2) Every entity class that is defined by an aggregation association with other classes without association types must have (or inherit) a user-specified key.

The constraints discussed above are those constraints that are implicitly or explicitly specified in the structural definition of a knowledge base. They are frequent constraints that can be more conveniently specified by users in the structural declaration of a knowledge base. Having enumerated the integrity constraint rules from the S-diagram, it is appropriate to classify such rules and discuss their role in query optimization.

Integrity Constraint Rule Classification

In many current studies, integrity constraint rules are classified according to their data dependence/independence [RAS88]. While such a classification does indicate when such rules can be considered for query optimization (a priori or run-time), it is not a direct indication of a rule's potential to optimize a query. In the following section, an alternative classification scheme is presented using the terminology of Shenoy, Ozsoyoglu, and Siegel [SHE87] and [SIE88]. This scheme distinguishes transition integrity constraint rules, which perform operations to maintain a knowledge base's semantic correctness, from state integrity rules, which specify semantically valid/invalid knowledge base states. As will be shown later, state rules, are more directly beneficial to query optimization

than transition rules. However, these two categories are not incompatible in that both meet the integrity constraint rule definition, and rules from one category can be directly converted to the other. Though Shenoy and Ozsoyoglu [SHE87] further classifies state integrity constraints, the finer distinction is not necessary in OSAM*'s context of query optimization using rules.

Transition Integrity Constraint Rules

As stated, transition integrity constraint rules maintain a knowledge base's semantic correctness by performing operations when triggered by other operations or by a knowledge base state. The operations that the rule performs are the actions that will maintain the knowledge base's integrity. Consider again the S-diagram of the university model in Figure 1. Assume that our knowledge base should ensure that every teacher teaches a section. Though not labelled as such in the diagram, this constraint implies total participation of the constituent class Teacher in the Section class. Rule 24 is a transition integrity constraint rule to represent this knowledge.

Rule 24:
IF Teacher ! Section
THEN DELETE Teacher.

Explanation: This rule implies that if a teacher is found that does not teach a section, then the teacher should be deleted from the knowledge base.

While the above rule is great for automatic integrity maintenance, its semantics are indirect. The ambiguity is because of the consequent action. It

does not define a valid or invalid state but rather performs an operation (the deletion of teacher objects not associated with section objects). The intention of the consequent operation must therefore be deduced. In this case, the rule is intended to never allow a teacher to not teach a section. While many integrity constraint rules are better defined as state rules, there are a few cases in which the semantics of the application fit the transition type of integrity rules better.

A case in point is the set equality constraint shown in Figure 5 of the Geometric and Machining classes. The set equality constraint specifies that these two classes represent two different perspectives of the same set of objects. Transition integrity rules keep the two classes consistent in an automatic and natural manner. When an object is altered in one class, the rule will be triggered and do the same changes to the other class. An transition rule will keep the classes equal without user interaction.

Another example where an transition rule is applicable is with the interaction association type. If an object of an interaction association is deleted then the other objects of the iteration should be deleted as well. For example, in the university schema again, if a student is deleted from the knowledge base, then all the associated transcript instances should be deleted as well, thus maintaining referential integrity. A transition rule can perform this operation automatically.

State Integrity Constraint Rules

As stated above, state integrity constraint rules explicitly specify valid and invalid knowledge base states. When triggered, the state rule checks to determine if an operation will or has violated the knowledge base's correctness; in which case, the operation is rejected, undone, or the user is informed of the violation. Two example state rules that are counterparts to Rule 24 are:

Rule 25:
IF Teacher THEN Teacher * Section.

Rule 26:
IF Teacher ! Section THEN NOT_EXIST Teacher.

Explanation: Rule 25 signifies that there must be an association between a teacher instance and a section instance. Rule 26 states that there is never a teacher who does not teach a section. Note that both rules do not perform any operation, but rather represent the valid and necessary associations of the knowledge base. They also allow a section to exist without a teacher, which is the original, intended meaning in this case.

The reason for distinguishing transition and state integrity constraint rules is two-fold. The first reason being that state rules should represent the majority of a knowledge base's integrity constraints. State rules explicitly define the semantics intended for the rules. The semantics of transition rules' consequent operation, however, are not directly defined. While the above integrity constraint rules that are mapped from the S-diagram examples could be

designated as transition rules, their semantics are better represented by the given state rules. The second reason for the transition/state integrity rule categorization is to illustrate how state rules are directly usable to query optimization, while the semantics of transition rules must be deduced before they can be used in optimization. Suppose a simple query is entered to request those teachers who do not teach a section. While Rules 24 through 26 are intended to have the same meaning, state Rules 25 and 26 logically indicate that no such teacher exists. Rule 24, however, is not so direct. One must first deduce that since teachers who do not teach a section have been automatically deleted, there will not be an instance of a teacher not teaching a section in the knowledge base. In making such a deduction, one is conceptually converting the transition rule into a state rule; which in turn, answers the query. In short, a KBMS needs such a mechanism if it is to make full use of transition as well as state integrity constraint rules in query optimization. Because such a conversion will take additional time, the majority of the integrity constraints should be state rules.

In summary, Figure 9 is a hierarchial diagram of the knowledge rule categories. The distinction between the types makes it apparent that state integrity constraint rules offer the highest query optimization potential; that is, they are most likely to be beneficial in improving query processing. Deductive rules have variable savings potential (the savings may be great or insignificant

depending on the particular amount of data being deduced) and transition integrity rules require extra interpretation to extract their exact semantics. Therefore, these last two rule types are not as desirable to query optimization. Thus, the next chapter focuses on state integrity constraint rules and the exact methodology to use them in query optimization.

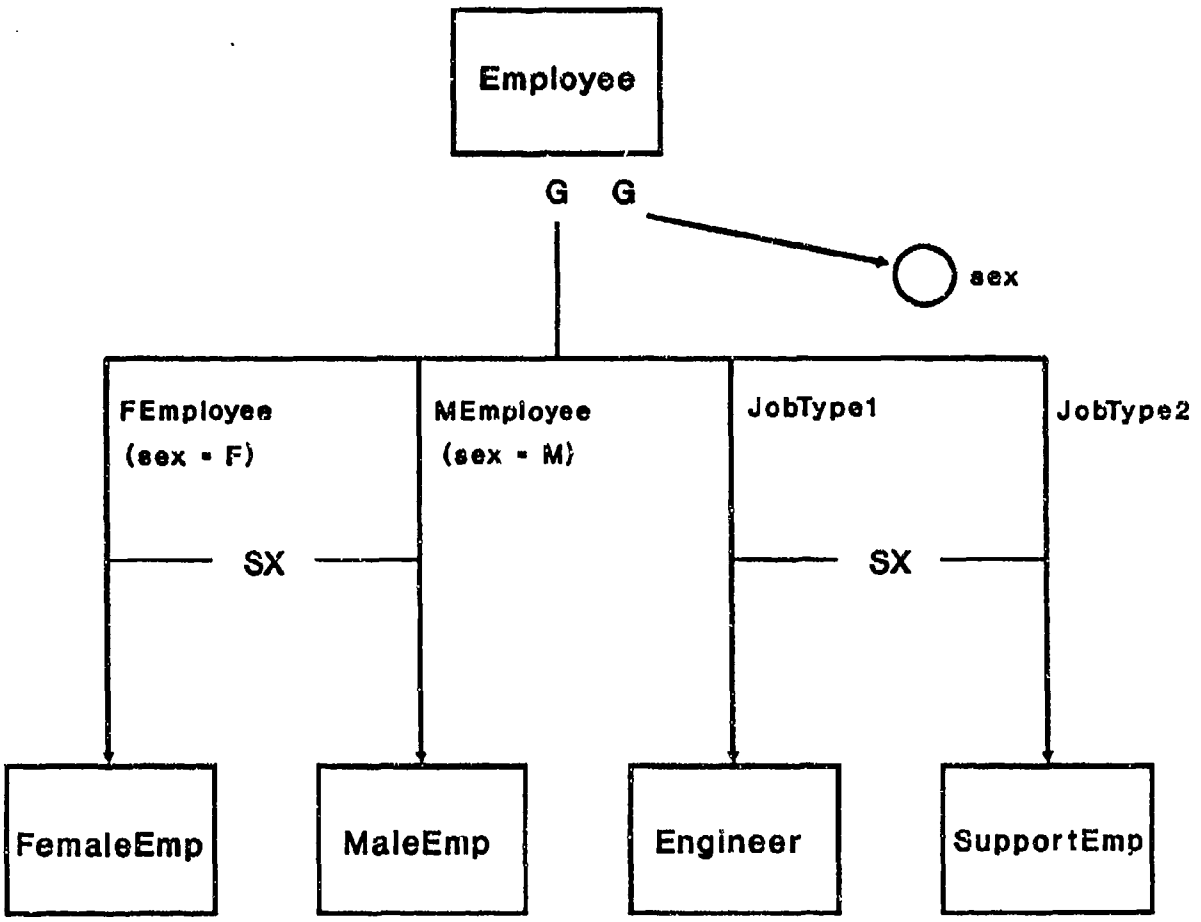

Figure 4. Employee Database Schema.

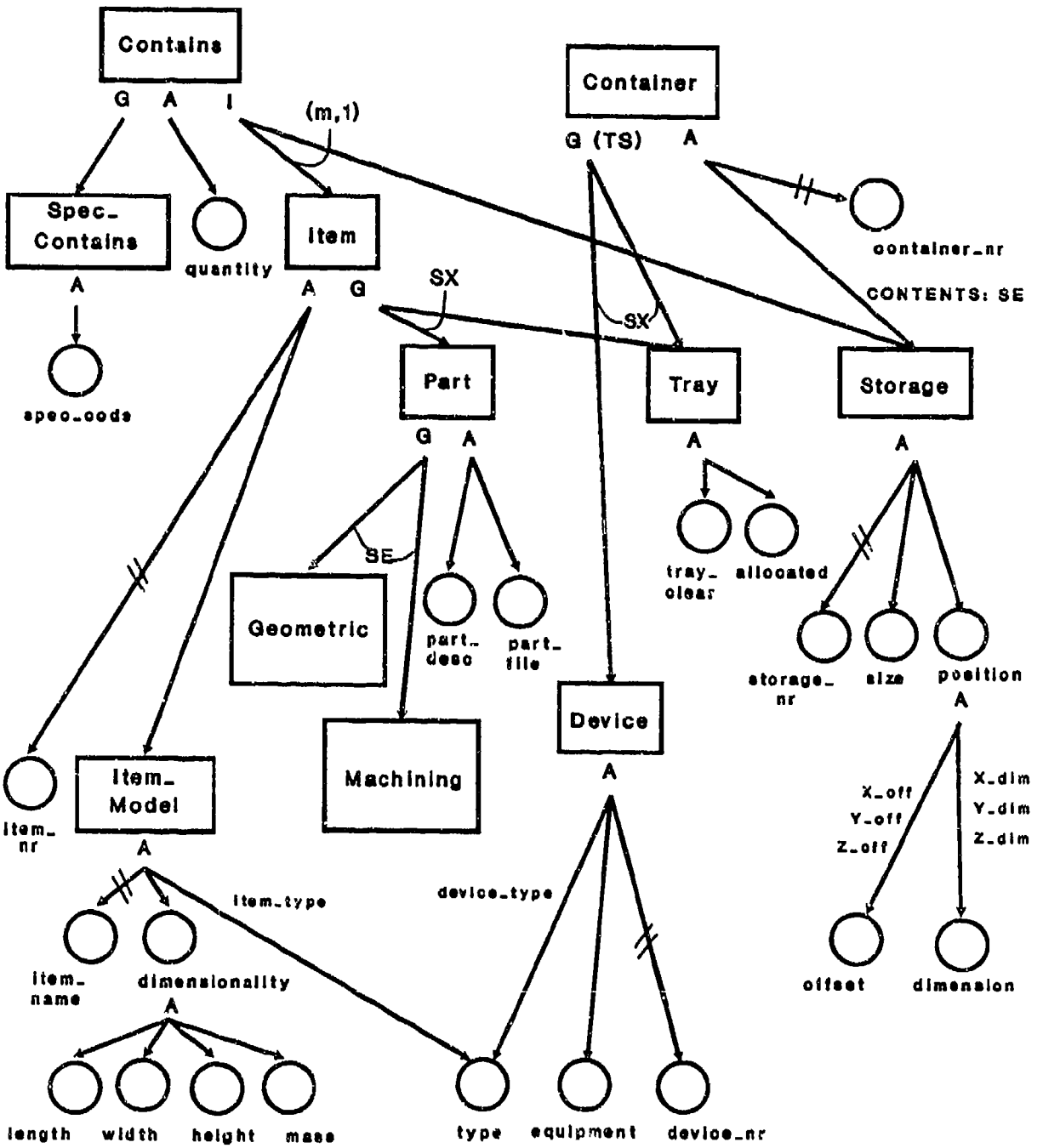

Figure 5. Manufacturing Database Schema.

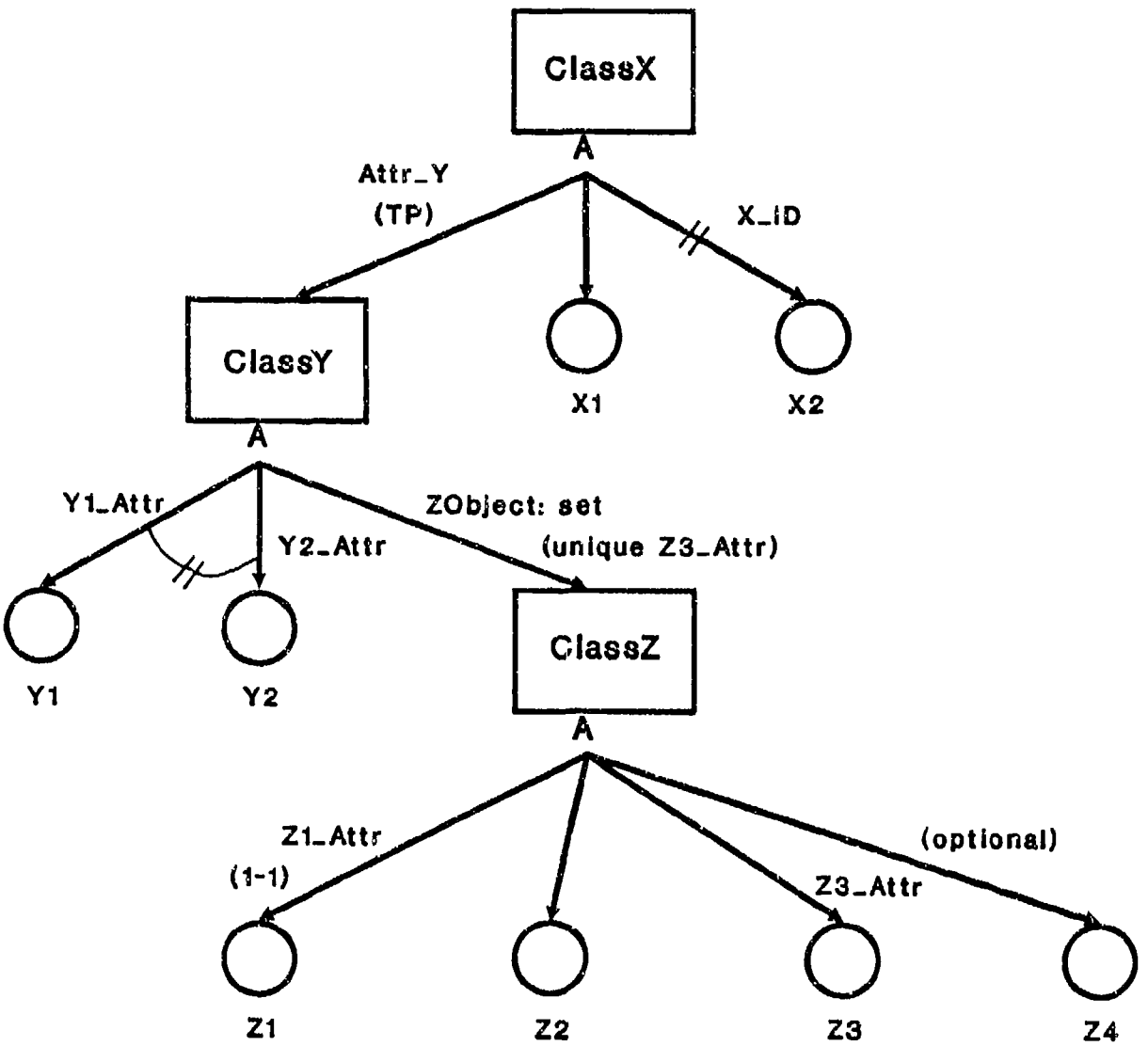

Figure 6. Generic Database Schema.

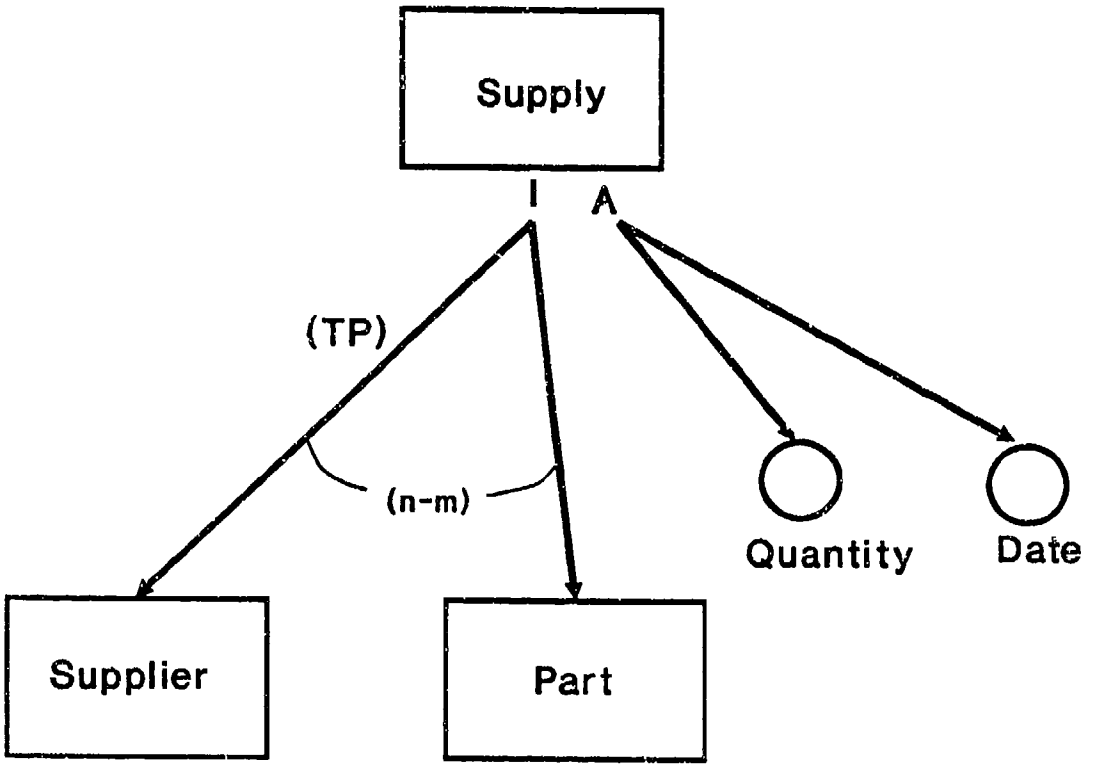

Figure 7. Supply Database Schema.

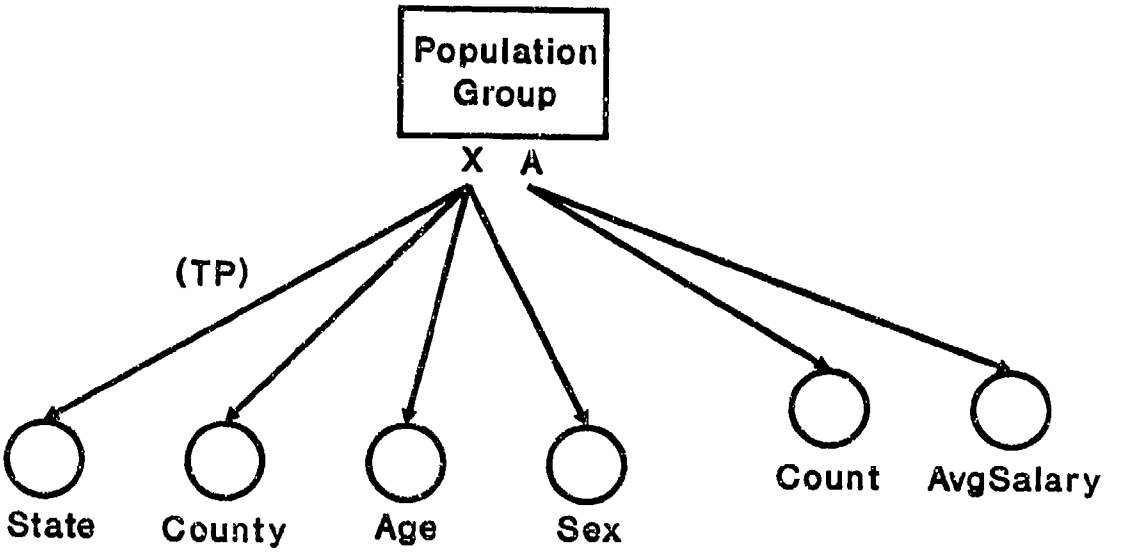

Figure 8. Population Database Schema.

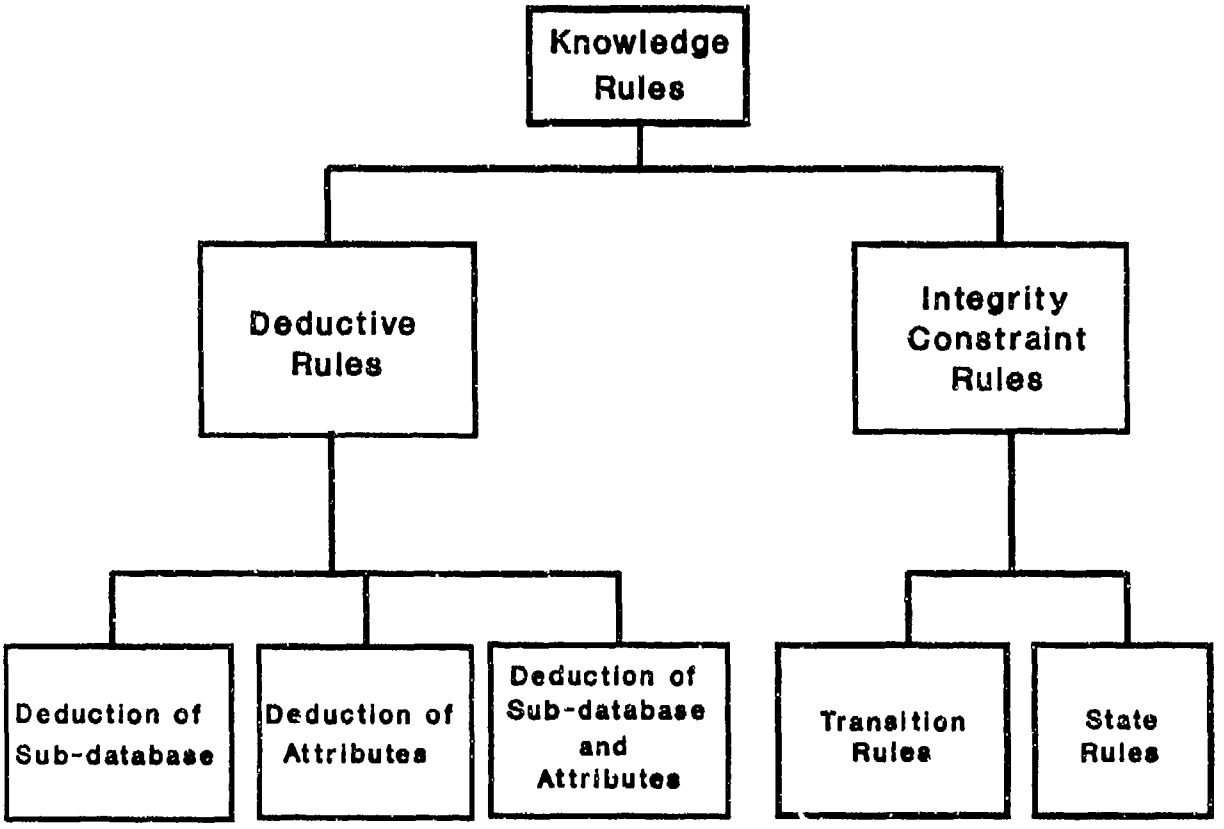

Figure 9. Knowledge Rule Classifications.

CHAPTER 5
SEMANTIC QUERY OPTIMIZATION IN OSAM*

After an intensive study of semantic query optimization in OSAM*'s KBMS, a set of transformation rules to perform optimization has been developed. This chapter serves two purposes. Firstly, it defines the transformations rules with examples. Secondly, it serves to exemplify the applications of these transformation rules through a series of case studies. For each case study, a query contrived to exemplify a certain aspect of the transformation rules is presented in its initial form. The transformation rules are then applied to generate alternative query execution plans. For each transformation step, the transformation rule applied and the constraint rule used are identified. The case studies demonstrate the step-by-step conversion of a query to a point where an answer can be returned or all pertinent rules have been applied. This chapter defines and demonstrates the complete set of transformation rules utilized to accomplish semantic query optimization in OSAM*.

Though this chapter defines valid semantic query transformation rules, it does not consider the criteria for a specific transformation. This topic is discussed in the next chapter. It is sufficient to emphasize now that the

alternative plans generated by the transformation rules may or may not be more efficient than executing the query in its initial form. Along the same lines, note that each transformation step (each time a rule is used to generate a different context), an alternatively valid execution plan is created. Many of the case studies go through several transformations before completion. For each iteration, the alternative generated must be considered a valid plan and must be analyzed to determine if it is more efficient than all the other plans generated for that query. The "profitability" of a query execution plan is discussed in the next chapter.

Transformation Rules For Semantic Query Optimization

This section defines the transformation rules necessary to perform semantic query optimization in OSAM*'s KBMS. These rules are the result of a study on employing every type of knowledge rule in the optimization process. The use of state integrity constraint rules becomes the main focus of the study since they are the most beneficial to the optimization process, as discussed in Chapter 4. All sub-types of state integrity constraint rules were studied for the development of the transformation rules. In this manner, the set of transformation rules is believed to be complete.

Transformation Rule 1: Reduction to Binary Associations

The association patterns of a query's context expression and of a rule's antecedent and consequent should be reduced to a series of binary associations.

This transformation step is imperative for the definition of other transformation rules. It allows for a conceptually simpler and clearer definition of the remaining transformation rules by standardizing the association pattern format. For example, (A * B * C • A * B * D) reduces to the AIntersect expression: (A * B) • (B * C) • (A * B) • (B * D). Let's examine the validity of this process by the example object diagram in Figure 3. Following the proper order of precedence, the object associations that satisfy A * B * C are (a1b1), (b1c3), (a2b4), (b4c2), while the object associations that satisfy A * B * D are (a2b4), (b4d1), (a4b5), (b5d2). The intersection of these patterns then is (a2b4), (b4c2), (b4d1), as highlighted in the diagram. Alternatively, evaluate the pattern using only binary associations. Object associations satisfying A * B are (a1b1), (a2b4), (a3b2), (a4b5); satisfying B * C are (b1c3), (b3c1), (b4c2); and satisfying B * D are (b4d1), (b5d2). Next the AIntersect of (A * B) • (B * C) produces (a1b1), (b1c3), (a2b4), (b4c2), while (A * B) • (B * D) produces (a2b4), (b4d1), (a4b5), (b5d2). Finally, intersecting these results produces the same answer (i.e., (a2b4), (b4c2), (b4d1)) as what is derived from the non-binary association pattern.

The reason for reducing association patterns to binary relationships is two-fold. First, it simplifies the definition of transformation rules. The rules are easier to define once the patterns are standardized into binary associations. The semantic converse rule (Transformation Rule 5) and the logical contradiction rule (Transformation Rule 6) would be much more complicated to define if it is

not assumed that the association patterns are a series of binary relationships. Secondly, simplification of context expressions becomes easier. Logical contradictions and redundancies become more apparent when the patterns are reduced to a series of binary associations. As in the above example of (A * B * C • A * B * D), the pattern is converted to (A * B) • (B * C) • (A * B) • (B * D). In this form it becomes apparent that the (A * B) is redundant and one instance can be removed to simplify the expression to (A * B) • (B * C) • (B * D). Finally, distributions may have to be performed across parenthesized sub-expressions in order to perform the reduction to binary associations. The distributive properties of all association operators are defined by Guo, Alashqur, Lam, and Su [GUO89].

Transformation Rule 2: Semantic Expansion

This rule pertains to state integrity constraint rules and is stated in Chapter 3 as follows: If the antecedent of a rule is satisfied by the constraints within a query's context, then the rule's consequent can be incorporated as another restriction to the query, provided that restriction is not already present in the query. The consequent is concatenated to the query's context with an AIntersect operator, "•". There are several sub-rules falling under this rule type depending on whether the antecedent of the state integrity rule or the context of the query have attribute restrictions within them.

a. Semantic expansion when no attribute range is specified in the query's context or in the constraint rule's antecedent: The association pattern of a rule's antecedent is satisfied if that pattern is encompassed by a broader or equivalent association pattern specified in a query's context. For example, if a rule's antecedent is (B * C) and a query has the context of [A * B * C * D], which equals [(A * B) • (B * C) • (C * D)], then the rule's antecedent is satisfied and its consequent can be added as an additional restriction.

b. Semantic expansion when an attribute range is specified in the query's context: In this case, an attribute range may or may not be specified in the antecedent of a constraint rule. If the context of a query restricts a particular attribute to a set (or range) of values, then a rule's antecedent must pertain to a super-set (or greater and encompassing range) of those attribute values in order for the rule's antecedent to be satisfied. For example, if a the context of a query is (A[a1 < 5]) and a rule's antecedent is (A[a1 <= 7]), then the rule may be used for semantic expansion, as its antecedent is satisfied by the query's context.

c. Semantic expansion when an attribute range is specified in a rule's antecedent but not in a query's context: If a constraint rule's antecedent contains an attribute range and the query's context does not contain a restriction on the same attribute, then semantic expansion cannot be performed. The rule's antecedent is not necessarily satisfied. For example, if the context of a query is

(A) and a rule's antecedent is (A[a1 <= 2]), then expansion using this constraint rule cannot be done. It is quite possible that there exists object instances in class A where attribute a1 is greater than 2. The rule's consequent does not apply to those instances. In such situations, the consequent cannot be added as an additional restriction to the context of the query.

Transformation Rule 3: Semantic Replacement

This transformation rule is much like the semantic expansion rule (Transformation Rule 2), except that it pertains to deductive knowledge rules. The goal of this rule is not to expand the query with an additional restriction, but rather to replace some or all of a query's context with the consequent of the deductive rule (i.e., the derived data). The replacement is feasible only when the derived sub-database is materialized prior to query processing, for it would be highly inefficient to derive data during processing when compared with executing the original query without data derivation. Semantic replacement is defined as follows:

If the antecedent of a deductive rule is satisfied by the constraints within the context of a query, then the rule's consequent can replace those restrictions that satisfy the antecedent, provided the sub-database contains the necessary attributes to uphold the remaining attribute restrictions in the context. To determine whether the context satisfies a rule's antecedent, use the same criteria as that specified for semantic expansion (Transformation Rule 2, sub-rules a

through c). Additionally, the deductive rule must project the necessary attributes into the resultant sub-database to uphold all attribute restrictions not replaced in the transformation. For example, if a query's context is (A[a1 > 25] * B * C) and a deductive rule is IF A * B THEN A-B(B[b1, b2]), then semantic reduction cannot be done as the (a1 > 25) attribute restriction cannot be represented in the derived A-B sub-database. The a1 attribute is not retained in the sub-database A-B. If the deductive rule is IF A * B THEN A-B(A[a1], B[b1, b2]), then semantic replacement could take place to produce a query context of (A-B[a1 > 25] * C).

Transformation Rule 4: Semantic Reduction

This rule pertains to state integrity constraint rules as follows: If the constraints specified in a query's context imply the antecedent of a rule and the rule's consequent is also implied by a restriction already present in the query, then that restriction is redundant and may be removed, provided its removal will not result in an empty query context. Implication of a rule's antecedent and consequent must meet the same satisfaction requirements as those outlined in Transformation Rule 2, sub-rules a through c. Though these sub-rules pertain to a rule's antecedent, they are equally valid and applicable to satisfying a rule's consequent.

In some cases, application of semantic reduction could lead to removal of the entire context expression from a query, since the entire context may be

flagged as a redundant restriction. This situation arises when the entire context of the query implies the consequent of the rule (in addition to the rule's antecedent being satisfied by the context). Obviously, total removal of the context is not allowed because the query must have some specification as to what portion of the database it is to be applied against. In such cases, the portion of the context (i.e., the sub-expression) that satisfies the rule's antecedent must be retained. If more than one sub-expression of the context satisfies the rule's antecedent, then any of the satisfying sub-expressions may be retained.

Transformation Rule 5: Semantic Converse Rules

When a user defines a state integrity constraint rule, there also exists what is termed a "semantic converse rule." The semantic converse rule is the complement of a state integrity constraint rule. In general, if the rule's structure is given as IF A THEN B, then its converse is IF NOT(B) THEN NOT(A), where A and B are the antecedent and consequent of a state integrity constraint rule, respectively, and NOT(B) and NOT(A) represent complement association patterns. Assuming the knowledge base is constantly in a valid state (i.e., there exists no instances in the knowledge base that violate a state integrity constraint rule), semantic converse rules are valid rules that can be used for semantic query optimization; just as any other state integrity constraint rule. The validity of this assertion is ascertained by perceiving a rule in a cause and effect manner:

view the antecedent of a rule as the cause of an effect, where the effect is the consequent of the rule. Then, a converse rule implies that if the effect does not exist, then the cause of the effect cannot possibly exist.

Thus, for every state integrity constraint rule, there actually exists two rules that can be used for optimization: the rule itself and the semantic converse rule of that rule. The complexity of generating a semantic converse rule, however, can be restrictively large, depending on the complexity of the association patterns used in the integrity constraint rule. For example, it is clear that the complements of association patterns such as (A * B) and (A[a1 < 6]) are (A ! B) and (A[a1 >= 6]), respectively. For an association pattern with more classes and attribute restrictions, however, the generation of converse rule becomes much more complicated and lengthy. For example, the association pattern (A * B * C * D) has a complete converse association pattern of [(A ! B * C * D) + (A ! B ! C * D) + (A ! B ! C ! D) + (A * B ! C * D) + (A * B ! C ! D) + (A * B * C ! D) + (A ! B * C ! D)]. Specifically, if there exists a total of n association operators and attribute equivalence operators in an association pattern, then an expression with $(2^n - 1)$ AUnion operands will be generated for a complete converse association pattern.

To avoid generating large converse rules, which in turn would lead to large overheads in the optimization process, a simple methodology has been developed to create simpler converse rules. This methodology generates

completely valid state integrity constraint rules that are constructed from a sub-set of the total possible converse sub-expressions. This methodology limits its scope by focusing on binary associations constructed in Transformation Rule 1. The methodology to construct the simplified converse rules is as follows:

a. Association operators are replaced accordingly: "*", "!", "•", "+" are replaced by "!", "*", "+", and "•" respectively.

b. Attribute equivalence operators are replaced by their converses, meaning "=", "!=", "<", ">", ">=", "=<" are replaced by "!=", "=", ">=", "=<", "<", and ">" respectively.

c. Logical AND operators are replaced by OR operators. Logical OR operators are replaced by AND operators.

d. Existence operators are replaced by their converses. Association patterns consisting of a single class with no association or attribute restriction are implied to have a default "EXIST" operator (if it is not explicitly stated) which should be replaced by a "NOT_EXIST" operator. Likewise, "NOT_EXIST" operators of association patterns consisting of a single class with no association or attribute restriction should be replaced by "EXIST" operators.

e. The rule configuration is reversed. That is, the rule's antecedent becomes the consequent, and the rule's consequent becomes the antecedent.

Note that this methodology does not imply that the converse of the AIntersect (•) operator is AUnion (+), nor is the OR operator the converse of

the AND operator. Rather, the instruction to replace each with the other is part of the strategy to limit the scope of generating full complement association patterns. For example, the full complement expression of the association pattern (A * B) • (B * C) is very complex and involves association operators not introduced in this thesis. The methodology above suggests that (A ! B) + (B ! C) is a simpler (but not complete) compliment association pattern. For example, in Figure 3, (A * B) • (B * C) produces (a1b1), (b1c3), (a2b4), (b4c2). A full complement expression to this expression would identify all other patterns that do not contain these specific associations. The expression (A ! B) + (B ! C) produces a sub-set of such patterns, specifically: a5, b3 and b2, b5, c4 from the (A ! B) and (B ! C) operands, respectively. The a3 and c1 objects are missed by this simplified, complement association pattern. The validity of replacing AUnion/AIntersect and AND/OR operators as specified thus rests on the fact that the other association operators (i.e., the Associate/NonAssociate operators) and the attribute equality operators are truly complemented. The truly complemented sub-expressions are either AUnioned or AIntersected to generate an expression that is less lengthy and complex than the full complement expression.

In summary, the converse rules enhance semantic query optimization by generating more state rules that can be considered for query transformations.

The length and complexity of the converse rules is purposely limited to avoid large overhead in query processing.

Transformation Rule 6: Context Simplification

After each semantic expansion or replacement, a query's context expression must be checked for simplification. The context statement can be simplified in one of several ways:

a. Logical contradiction. After a semantic expansion or replacement, a logical contradiction may exist between an existing sub-expression in the context and the newly added restriction. Sub-expressions of a query's context are those portions of the association pattern that are separated by the AIntersect operator(s). The logical contradiction may exist in an association between classes, such as (A * B) • (A ! B), or in restrictions on an attribute, such as (A[a1 > 25]) • (A[a1 < 15]), or in a combination of both, such as (A[a1 > 25] * B) • (A ! B). To detect a logical contradiction, the restriction added by semantic expansion/replacement must be compared with all existing sub-expressions in the context. The restriction contradicts a sub-expression if it restricts any class to a set of object instances that is logically exclusive to a set of object instances meeting the restriction of an existing sub-expression.

b. Simplification of an AUnion (+) expression. After a semantic expansion or replacement, an AUnion expression can be simplified because of the additional restriction to the context. If the restriction added to a query's

context is equivalent to an operand of an AUnion expression in the context, then the AUnion expression can be simplified to consist of only that operand. That operand, in turn, can also be eliminated since it is redundant to the restriction added to the context expression of the query. Overall, the entire AUnion expression is replaced by the added restriction. On the other hand, if an added restriction contradicts an operand of an AUnion expression in the query's context, then that operand can be eliminated from the AUnion expression.

To exemplify these two particular cases, consider the object diagram in Figure 3. For the first case, suppose the query context (A * B * C) + (A * B * D) undergoes semantic expansion to add the restriction (A * B * C), thus making the context expression (A * B * C) + (A * B * D) • (A * B * C). The AUnion operator, according to this sub-rule, can be simplified so that the remaining context is (A * B * C). Specifically, in Figure 3 the expression (A * B * C) produces (a1b1), (b1c3), (a2b4), (b4c2), while (A * B * D) produces (a2b4), (b4d1), (a4b5), (b5d2). Therefore the AUnion of these two operands is (a1b1), (b1c3), (a2b4), (b4c2), (b4d1), (a4b5), (b5d2). If this result is then AIntersected with (A * B * C), the same association set as that of (A * B * C) results: (a1b1), (b1c3), (a2b4), (b4c2). To exemplify the second case of this sub-rule, suppose the context of a query (A * B) + (B * D) undergoes semantic expansion to add the restriction (B ! D), thus making the context expression (A

* B) + (B * D) • (B ! D). According to this sub-rule, the (B ! D) restriction contradicts the (B * D) AUnion operand, so that the expression can be simplified, making the context (A * B) • (B ! D). Again in Figure 3, (A * B) produces (a1b1), (a2b4), (a3b2), (a4b5), while (B * D) produces (b4d1), (b5d2). The AUnion of these two operands is (a1b1), (a2b4), (a3b2), (a4b5), (b4d1), (b5d2). When this result is the AIntersected with (B ! D), which produces b1, b2, b3, d2, d4, the result is the association pattern (A * B) • (B ! D), which produces the association set of (a1b1), (a3b2).

c. Attribute range simplification. Semantic expansion or replacement may introduce an attribute range restriction to a context expression that already restricts that same attribute. One of three situation will arise at such an occurrence: the attribute restrictions will contradict each other, the range of one attribute restriction will wholly overlap the other, or the attribute ranges may partially overlap each other. In the first case, the logical contradiction exists and a null answer must be returned. For example, if the restriction (A[a < 5]) is added to the context expression (A[a >= 6] * B) of a query, then a contradiction exists. A null answer is the result of the query. This case is pertinent to sub-rule a of this transformation rule. In the second case, the attribute restriction that wholly overlaps the other attribute restriction may be eliminated. If the context expression (A[a >= 6] * B) undergoes semantic expansion/replacement and the restriction (A[a > 20]) is added, then the context

expression must become (A[a > 20] * B) since the restriction with the narrower range (i.e., (a > 20)) must prevail over the other. In the final case of partially overlapping attribute restrictions, the restrictions can be combined to represent an intersection of the attribute range restrictions. For example, if the context expression (A[a >= 6] * B) of a query has the restriction (A[a < 20]) added to it, the attribute restrictions can be combined to form a simplified context with only one attribute range (A[6 <= a < 20] * B).

d. Association algebra simplifications according to commutative, associative, reflexive, and distributive properties. In [GUO89], the mathematical properties of association algebra operators are given. They can be used to simplify the context statements of a query. For example, an expression such as A • A * B can be simplified to A * B according to the identity property of association algebra.

Transformation Rule 7: Singularity of Rules

This transformation rule pertains more to rule definition than to transformation methodologies. A single semantic property must be written as a single rule. Violation of this rule does not threaten the integrity of the knowledge base, but restricts the rule's usefulness to semantic query optimization. If the full semantics of a property are not enveloped in a single rule, then the separate rules making up that semantic property may not be, in and by themselves, applicable to transformations. The KBMS would then need

a mechanism to combine such rules and additional overhead would be incurred for query processing. Such a mechanism and overhead could be avoided if every semantic property is expressed as a single rule. This rule is strongly exemplified later in Case Study 4.

Transformation Rule 8: Automatic Generation of Rules Representing Derivation Knowledge If a derived sub-database is referenced in a query, then the knowledge of how the data in the sub-database is derived can be represented in a rule which can be considered for transformations. In order to generate such a rule, however, the condition(s) on which the data is derived must be expressible in terms of the attributes in the derived sub-database. For example, the deductive rule IF A[a1 > 5] THEN Large-A(A[a1,a2]) can be used to generate the constraint rule IF Large-A THEN A[a1 > 5]. The attribute restriction of (a1 > 5) fully represents the condition on which data is derived for the Large-A sub-database. If the attribute restriction of the deductive rule's antecedent is instead (a1 > 5 OR a3 < 10), then the above constraint could not be generated, since the a3 attribute is not in the resultant sub-database and the (a1 > 5) restriction alone does not fully express a condition for deriving data into the sub-database. Finally, if the attribute restriction of the deductive rule is (a1 > 5 AND a3 < 10), then the (a1 > 5) restriction does fully express a condition (i.e., one of the two conditions) for data derivation and the above integrity constraint rule IF Large-A THEN Large-A[a1 > 5] can be realized.

Case Studies Exemplifying Transformation Rules

Having defined the transformation rules for semantic query optimization in OSAM*'s KBMS, case studies exemplifying the application of the transformation rules would be very beneficial to a clearer and deeper understanding of the rules. The following case studies are structured as follows: Within each case study, a short explanation of what the study intends to demonstrate is given first. Next, pertinent knowledge rules for the study are identified or defined. The appendix lists all the knowledge rules used throughout the thesis and serves as a quick, consolidated look-up of rules referenced in the case studies. For each pertinent rule, a rule implication is listed. It has the form: "rule antecedent --> rule consequent," meaning the antecedent of a rule implies its consequent. Next, a contrived query is stated (verbosely and in OQL syntax). The query is designed to illustrate specific transformations rules. Using the initial query context as a starting point, a series of transformation steps is performed whereby the information contained in the pertinent knowledge rules is used to generate semantically equivalent context expressions. Each transformation step is an application of one of the defined transformation rules. The transformation rule and the knowledge rule being used (if one exists) are identified with each query alteration. Finally, following the transformation steps, the resulting query plan, the overall transformation

process, and the case study is discussed. In total, the case studies cover the use of every type of integrity constraint state rules and the use of deductive rules.

Case Study 1

Figure 4 Schema. This case is an example of query transformations by semantic expansion that lead to a null query answer. It emphasizes the logical equivalence of rules, showing how rules can be stated differently but will lead to the same result. Specifically, this case illustrates that every rule has a semantic converse rule which is beneficial to query optimization. Finally, it illustrates how to detect logical contradictions in the context expression.

Pertinent Rules: Rules 8, 9, and 10.

Pertinent Rule Implications:

Rule 8:
Employee * Engineer --> Employee ! SupportEmp.

Rule 9:
Employee * SupportEmp --> Employee ! Engineer.

Rule 10:
(Employee * Engineer) • (Employee * SupportEmp) --> NULL.

Note: Transformation Rule 1 is applied to Rule 10's antecedent to reduce Employee's ternary relationship to two AIntersect operands with binary associations.

Query: "Display employees who are both engineers and supporting employees."

OQL:

```
CONTEXT Employee * AND(Engineer, SupportEmp)
        SELECT Employee[name]
DISPLAY
```

Query Transformation Using Rule 8:

Initial:

Employee * AND(Engineer, SupportEmp).

Step 1: Transformation Rule 1 is applied to reduce Employee's ternary relationship to an AIntersect expression with operands having binary associations. Thus, the resulting expression is:

(Employee * Engineer) • (Employee * SupportEmp).

Step 2: Transformation Rule 2, semantic expansion is applied to add the consequent of constraint rule 8 as an additional restriction to the query. Since a restriction does not exist in either the context expression or in the rule's antecedent, sub-rule a of Transformation Rule 2 is applied. The rule's antecedent is satisfied by the context expression according to this sub-rule. Thus, the resulting expression is:

(Employee * Engineer) • (Employee * SupportEmp) •
(Employee ! SupportEmp).

Step 3: Transformation Rule 6, context simplification, is applied to the context of the query to determine that a logical contradiction exists (sub-rule a) and a null answer must be returned. Specifically, the AIntersect operands of (Employee * SupportEmp) and (Employee ! SupportEmp) contradict. The context expression simply reduces to:

NULL.

Query Transformation Using Rule 9:

Initial:

Employee * AND(Engineer, SupportEmp)

Step 1: Transformation Rule 1 is applied to reduce Employee's ternary relationship to an AIntersect expression with two operands having binary associations. The context expression thus becomes:

(Employee * Engineer) • (Employee * SupportEmp).

Step 2: Transformation Rule 2, semantic expansion, sub-rule a, is applied to add the consequent of constraint Rule 9 to the context of the query, making the context:

(Employee * Engineer) • (Employee * SupportEmp) •
(Employee ! Engineer).

Step 3: The context simplification rule, Transformation Rule 6, is applied to determine that a logical contradiction exists. Specifically, the AIntersect operands of (Employee * Engineer) and (Employee ! Engineer) contradict. The query should return a null answer; the context simply becomes:

NULL.

Query Transformation Using Rule 10:

Initial:

Employee * AND(Engineer, SupportEmp).

Step 1: Transformation Rule 1 is applied to reduce Employee's ternary relationship to an AIntersect expression with operands having binary associations. Thus, the resulting expression is:

(Employee * Engineer) • (Employee * SupportEmp).

Step 2: Semantic expansion, Transformation Rule 2, is applied to add the NULL consequent of Rule 10 to the context of the query. Sub-rule a of Transformation Rule 2 is applied to add this restriction. Thus, the resulting expression is:

(Employee * Engineer) • (Employee * SupportEmp) • NULL.

Step 3: The context simplification rule (Transformation Rule 6) is applied to determine that the NULL restriction simplifies the AIntersect expression to simply NULL. The query must return a null answer. The context expression simply reduces to:

NULL.

Rules 8, 9, and 10 are logically related to each other in that any rule can be used to derive the other two rules. For example, Rule 8 not only states that employees who are engineers cannot be also supporting employees, but also all employees who are supporting employees cannot be engineers. This fact is apparent by examining Rule 8 in a cause and effect manner: An employee being instantiated as an engineer causes that employee not to be instantiated as a supporting employee. Conversely, if an employee is a supporting employee then he/she could not have been an engineer, else the effect would exist and would contradict with the statement that an employee is a supporting employee. That is to say, a cause cannot exist if the effect of the cause does not exist. As

an aside, the existence of an effect, however, does not imply the cause exists (the effect may be the result of another cause or causes). In essence, the converse statement above is the implication of Rule 9: an employee who is a supporting employee cannot also be an engineer. This reasoning can be extended a step further to show Rule 8 and 9 can be combined to derive Rule 10. In summary, logically equivalent rules can be used for query transformations to produce the same results.

Case Study 2

Figure 4 Schema. This case employs both semantic expansion and reduction transformation rules. It shows how a rule and its semantic converse rule must be considered in semantic expansion. This case also shows that a rule is pertinent if its antecedent is satisfied, but does not necessarily match query constraints exactly. Finally, this case also shows that an integrity constraint rule may be considered for transformations more than once for a given query.

Pertinent Rules: Rule 8 and the following rules:

Rule 27:
IF Employee THEN Employee * OR(Engineer, SupportEmp).

Explanation: This is a contrived total specialization constraint on the schema in Figure 4. It means that every employee must be an engineer or a supporting employee.

Rule 28:
IF Employee * Engineer THEN Employee.salary > $40k.

Rule 29:
IF Employee * SupportEmp THEN Employee.salary < $30k.

Explanation: Assume it is company policy that all employees who are engineers will make over $40k a year and all employees who are supporting employees will make less than $30k. State Rules 28 and 39 ensure these conditions are maintained in the knowledge base.

Pertinent Rule Implications:

Rule 8:
Employee * Engineer --> Employee ! SupportEmp.

Converse:
Employee * SupportEmp --> Employee ! Engineer.

Rule 27:
Employee --> (Employee * Engineer) + (Employee * SupportEmp).

Converse:
(Employee ! Engineer) • (Employee ! SupportEmp) -->
NOT_EXIST (Employee).

Note: Transformation Rule 1 is applied to reduce the ternary consequent of Rule 27 to an AUnion expression with two operands having binary associations for Rule 27's implication and to an AIntersect expression for Rule 27's converse implication.

Rule 28:
Employee * Engineer --> Employee.salary > $40k.

Converse:
Employee[salary <= $40k] --> Employee ! Engineer.

Rule 29:
Employee * SupportEmp --> Employee.salary < $30k.

Converse:

Employee[salary >= $30k] --> Employee ! SupportEmp.

Note: Transformation Rule 5 is applied to create all of the semantic converse rule implications given above.

Query: "Display all employees who make more than $35k."

OQL:

```
CONTEXT Employee[salary > $35k]
        SELECT Employee[name]
DISPLAY
```

Query Transformations:

Initial:

Employee.salary > $35k.

Step 1: Transformation Rule 2, semantic expansion, is applied to add Rule 27's consequent as a restriction to the context of the query. Since the context contains an attribute restriction, sub-rule b is used to determine if Rule 27's antecedent is satisfied by the initial context. Rule 27's antecedent addresses the entire Employee class and the query restricts the context to just those employees making more than $35k. Therefore, the antecedent does indeed pertain to a super-set of objects over those selected in the context of the query. Rule 27's consequent is therefore added as another restriction to the context, making the context:

(Employee.salary > $35k) •
[(Employee * Engineer) + (Employee * SupportEmp)].

Step 2: Semantic expansion is again applied to add the consequent of Rule 29's semantic converse rule. Sub-rule b of Transformation Rule 2 is again the appropriate sub-rule for the application. The context expression thus becomes:

(Employee.salary > $35k) •
[(Employee * Engineer) + (Employee * SupportEmp)] •
(Employee ! SupportEmp).

Step 3: After adding the restriction of (Employee ! SupportEmp) in step 2, the context expression can be simplified. Specifically, sub-rule b of Transformation Rule 6 can be used to simplify the AUnion expression of [(Employee * Engineer) + (Employee * SupportEmp)]. Since the added (Employee ! SupportEmp) restriction contradicts the (Employees * SupportEmp) operand of the AUnion expression, (Employee * SupportEmp) is no longer needed as an operand in the AUnion expression. It is eliminated and the AUnion expression simplifies to just the (Employee * Engineer) operand. Thus, the resulting expression is:

(Employee.salary > $35k) • (Employee * Engineer) •
(Employee ! SupportEmp).

Step 4: Transformation Rule 4, semantic reduction, can be accomplished at this point. Rule 8 is used to identify the redundancy in this case. Application of Transformation Rule 4 is detailed below:

"Constraints specified in a query's context imply the antecedent of a rule," i.e., (Employee * Engineer) in the context of the query --> (Employee * Engineer) in Rule 8's antecedent, "and the rule's consequent is also

implied by restriction already present in the query," i.e., (Employee ! SupportEmp) in the context of the query --> (Employee ! SupportEmp) in Rule 8's consequent, "then the restriction is redundant and can be removed."

Thus, the resulting expression is:

(Employee.salary > $35k) • (Employee * Engineer).

Step 5: Semantic expansion is applied to add Rule 28's consequent to the context. Sub-rule b is the appropriate sub-rule since the Employee class is restricted in the context to those employees having a salary over $35k. Rule 28's antecedent applies to the entire Employee class and therefore represents a super-set of objects over those meeting the query's restriction on the employee objects. The expansion transforms the context to:

(Employee.salary > $35k) • (Employee * Engineer) •
(Employee.salary > $40k).

Step 6: Transformation Rule 6 can once again be applied to simplify the context expression. According to sub-rule c of this transformation rule, the (Employee.salary > $40k) restriction represents a narrower range than the (Employee.salary > $35K) restriction. It therefore takes precedence and eliminates the (Employee.salary > $35k) restriction. Thus, the context expression is:

(Employee * Engineer) • (Employee.salary > $40k).

Step 7: Finally, semantic reduction (Transformation Rule 4) can be applied with consideration of Rule 28. Specifically, Rule 28's antecedent and consequent are given as restrictions in the context expression. The restriction matching the consequent (i.e., Employee.salary > $40k) is thus redundant and can be removed. The final resulting context expression is:

(Employee * Engineer).

Though there are more transformation steps involved in this case as compared to the previous case and a null answer does not result, the resultant query context is markedly different than the original query. The result may be very beneficial if data is clustered by engineers. On the other hand, the restriction on the salary attribute has been totally removed. Its removal may not be beneficial if the Employee class is indexed or sorted by the salary attribute. As such, the original context expression or that given in Step 6 would be more efficient to process. Determination of which plan is most efficient depends on underlying data structures and access paths.

In summary, this case has emphasized the importance of converse rules and simplification steps. It has demonstrated that a knowledge rule may be applicable more than once to a query's transformations. A costing plan should determine how many times a rule can be considered for transformations.

Case Study 3

Figure 4 Schema. This case illustrates how a very broad and general query can become very tightly restricted through semantic expansions. Tightly restricted queries result in smaller intermediate data sizes and thus are generally more efficient to process.

Pertinent Rules: Rules 8, 17, 28, and the following:

Rule 30:
IF Employee THEN Employee * OR(MaleEmp, FemaleEmp).

Explanation: Every employee of the Figure 4 schema must be a male or female employee. The above rule ensures that this specialization is maintained in the knowledge base.

Pertinent Rule Implications:

Rule 8:
Employee * Engineer --> Employee ! SupportEmp.

Converse:
Employee * SupportEmp --> Employee ! Engineer.

Rule 17:
Employee[sex == "M"] --> Employee * MaleEmp.

Converse:
Employee ! MaleEmp --> Employee[sex != "M"].

Rule 28:
Employee * Engineer --> Employee.salary > $40k.

Converse:
Employee[salary <= $40k] --> Employee ! Engineer.

Rule 30:
Employee --> (Employee * FemaleEmp) + (Employee * MaleEmp).

Converse:

(Employee ! FemaleEmp) • (Employee ! MaleEmp) -->
NOT_EXIST(Employee).

Note: Transformation Rule 5 is used to generate all of the above converse rule implications. Likewise, Transformation Rule 1 is used to reduce the ternary relationship defined in Rule 30's consequent to binary associations in an AIntersect expression.

Query: "Display all male engineer employees."

OQL:

```
CONTEXT Employee[sex == "M"] * Engineer
    SELECT Employee[name]
DISPLAY
```

Query Transformations:

Initial:

Employee[sex == "M"] * Engineer.

Step 1:

(Employee[sex == "M"] * Engineer) • (Employee ! SupportEmp).

Step 2:

(Employee[sex == "M"] * Engineer) • (Employee ! SupportEmp) •
(Employee * MaleEmp).

Step 3:

(Employee[sex == "M"] * Engineer) • (Employee ! SupportEmp) •
(Employee * MaleEmp) • (Employee[salary > $40k]).

The above transformation steps are the result of applying semantic expansions using Rules 8, 17, and 28, respectively. The query contained an attribute restriction from its initial definition. Therefore, sub-rule b of

Transformation Rule 2 is the appropriate sub-rule for determining if the constraint rules' antecedents are satisfied by the context of the query. Rule 17 is the only constraint rule that has an attribute restriction in its antecedent. Its attribute restriction is the same as that specified in the query. It is therefore satisfied by the query context. All the other constraints pertain to the entire Employee class and therefore represent a super-set of objects over those objects satisfying the query's attribute restriction.

Rule 30 is defined but never introduced into the context of the query. Though its antecedent is satisfied by the context, its consequent would be a redundant restriction if it were added to the expression. Since the restriction of (Employee * MaleEmp) already exists in the context, adding the AUnion expression [(Employee * FemaleEmp) + (Employee * MaleEmp)] (i.e., Rule 30's consequent) would be redundant. If this transformation had been performed, the very next step would result in the removal of that same AUnion expression (as prescribed in Transformation Rule 6, sub-rule b). Thus, before a restriction is added by semantic expansion, the context must be checked to ensure that the restriction will not be redundant.

Finally, in the evaluation of the transformation result, the context expression has many more restrictions than the original query. Depending on lower level details such as data structures and access paths, the additional restrictions can be very beneficial. For example, if the data is clustered by

engineers and supporting employees or by male and female employees, the transformed query context would be much more efficient.

Case Study 4

Figure 5 Schema. The first query in this case is another example of semantic expansion leading to a logical contradiction in a context expression and thus to a null answer. The second query illustrates how a semantic property should be written as a single rule. Rules not representing the full semantic property may not be applicable to query transformations while a single rule may be.

Pertinent Rules: Rules 11, 12, and 13.

Pertinent Rule Implications:

Rule 11:
Part * Geometric --> Part * Machining.

Converse:
Part ! Machining --> Part ! Geometric.

Rule 12:
Part * Machining --> Part * Geometric.

Converse:
Part ! Geometric --> Part ! Machining.

Rule 13:
(Part * Geometric) + (Part * Machining) -->
(Part * Geometric) • (Part * Machining).

Converse:
(Part ! Geometric) + (Part ! Machining) -->
(Part ! Geometric) • (Part ! Machining).

Note: Transformation Rule 5 is used to generate the converse rule implications given above. Likewise, Transformation Rule 1 is used to create the implications for Rule 13 in the standardized binary association format.

Query: "Display all parts that are geometry parts but not machining information parts."

OQL:

```
CONTEXT (Part AND(* Geometric, ! Machining)
        SELECT Part[item_nr,part_desc]
DISPLAY
```

Query Transformations:

Initial:

Part AND(* Geometric, ! Machining).

Step 1: Transformation Rule 1 is used to reduce the context expression to the following standardized binary association format:

(Part * Geometric) • (Part ! Machining).

Step 2: Semantic Expansion is applied to perform a transformation step, whereby Rule 11's consequent is introduced as an additional restriction to the context. The context expression thus becomes:

(Part * Geometric) • (Part ! Machining) •
(Part * Machining).

Step 3: According to Transformation Rule 6, a logical contradiction is produced in the context expression in Step 2 between (Part ! Machining) and (Part * Machining). The query should return a null answer and the context simply becomes:

NULL.

Query: "Display all parts that are either geometric parts or machining parts."

OQL:

```
CONTEXT Part * OR(Geometric, Machining)
    SELECT Part[item_nr,part_desc]
DISPLAY
```

Query Transformations:

Initial:

Part * OR(Geometric, Machining).

Step 1: Transformation Rule 1 is used to reduce the context expression to the following standardized binary associations format:

(Part * Geometric) + (Part * Machining).

Step 2: This next step in the transformation process is fairly involved. First of all, note that neither Rule 11 or Rule 12 could be used for semantic expansion, for neither of their antecedents are satisfied by the context of the query at this point. As an example, Rule 11's antecedent (i.e., (Part * Geometric)) is a tighter restriction than the AUnion expression making up the context of the query. There are objects that will meet the conditions of the context but not those of Rule 11's antecedent (i.e., objects identified by the (Part * Machining) operand). Thus, Rule 11's antecedent is not satisfied. Using the same reasoning, Rule 12's antecedent is not satisfied at this point either.

While Rules 11 and 12 are defined together to maintain the set equality constraint on the Geometric and Machining classes, neither rule in and by itself

ensures this integrity constraint. It does not mean that these rules are invalid integrity constraints, for as long as both rules are present in the knowledge base, the set equality constraint is maintained. The problem of not defining this single semantic property as a single rule is pertinent to semantic query optimization. The multiple rules representing the single semantic property may not be applicable to a transformation, while a single rule encapsulating the full semantic property may be applicable. This case study, which uses Rules 11, 12 and 13, demonstrate this point. Thus, Transformation Rule 7 has been defined avoid the occurrence of this problem.

Rule 13, a rule that does encapsulate the full set equality constraint in a single rule, can be employed in the transformation process to perform semantic expansion (Transformation Rule 2). Its antecedent is exactly that of the context expression, permitting its consequent to be added as an additional restriction. The resulting context expression therefore is:

(Part * Geometric) + (Part * Machining) •
(Part * Geometric) • (Part * Machining).

Step 3: Transformation Rule 6, sub-rule b is applied to eliminate the AUnion expression in the context. Either of the sub-expressions added in Step 2 (i.e., (Part * Geometric) or (Part * Machining)) can be used to eliminate the Aunion expression. The resulting context is:

(Part * Geometric) • (Part * Machining).

Step 4: Either Rule 11, 12, or 13 can be used in a semantic reduction (Transformation Rule 4) at this point. Rule 13 is selected because it introduces an interesting scenario. Specifically, it demonstrates the possibility of losing the entire context expression in a semantic reduction if precautions are not taken. Rule 13's antecedent is satisfied by either operand of the AIntersect expression resulting from Step 3 (i.e., (Part * Geometric) or (Part * Machining)). Rule 13's consequent, however, is satisfied only be the entire context expression. Normally in a semantic reduction, the restriction satisfying the rule's consequent is eliminated, for it is redundant. In this case, however, doing so would leave an empty context expression. To avoid this precarious result, Transformation Rule 4 stipulates that the sub-expression satisfying the rule's antecedent must be maintained. Since either (Part * Geometric) or (Part * Machining) sub-expression satisfies Rule 13's antecedent, either can be retained as the context expression. The choice is arbitrarily made in this example to result in the following context expression:

(Part * Geometric).

In summary of this case study, some peculiar aspects of semantic expansion and reduction have been demonstrated. These aspects are compensated for in the transformation rules' definitions.

Case Study 5

Figure 1 Schema. This case illustrates the use of a deductive rule in a semantic replacement. In the first example of this case, a smaller sub-database replaces a large class specified in the original context of the query. The second example illustrates the use of the knowledge on how data is derived (as represented in rule format) to perform a semantic expansion. The semantic expansion then leads to a null answer in this example.

Pertinent Rules: Rule 19 and the following deductive rule:

Rule 31:
IF Course[c# >= 5000] THEN Grad_Courses[c#, college].

Pertinent Rule Implications:

Rule 19:
Section --> Section * Course.

Rule 31:
Course[c# >= 5000] --> Grad_Courses[c#, college].

Query: "Display the course numbers of graduate courses whose course number is less than 6000."

OQL:

```
CONTEXT Section * Course[5000 =< c# < 6000]
        SELECT Section[section#]
DISPLAY
```

Query Transformations:

Initial: Assume graduate course are known to have course numbers (c#'s) greater than or equal to 5000. The restriction on the c# attribute in the initial

context expression is then (5000 =< c# < 6000). Thus, the initial context expression is given as:

Section * Course[5000 =< c# < 6000].

Step 1: Rule 19 could now be employed in a semantic expansion (i.e., it's antecedent is satisfied by the context expression), except its consequent is a restriction already present in the context. Therefore it simply serves to verify the semantic validity of the query. Transformation Rule 3, semantic replacement, however, can be applied to transform the context. The Grad_Courses sub-database of Rule 31 can replace the Course class in the context expression. Since the attribute restriction of (c# < 6000) is not in Rule 31's antecedent, it must be presented as a restriction against the Grad_Courses sub-database in the transformed context expression. Had c# not been projected into the sub-database, this transformation step could not have been performed. Thus, semantic replacement results in the following expression:

Section * Grad_Courses[c# < 6000].

The sub-database that Rule 31 creates is a sub-set of instances of the Course class. It will therefore be more efficient to process the query against the sub-database since the amount of data being worked with will be less, provided the sub-database is materialized prior to query processing (as stipulated in Transformation Rule 3).

The next example shows that the knowledge on how data is derived is useful in query optimization as well. Rule 32 represents such knowledge:

Rule 32:
IF Grad_Courses THEN Grad_Courses[c# >= 5000].

Rule 32's Implication:

Grad_Courses --> Grad_Courses[c# >= 5000].

Note: Rule 32 should automatically be generated from Rule 31 (i.e., should not have to be user-defined) as prescribed in Transformation Rule 8.

Query: "Display all graduate courses with course numbers less than 5000."

OQL:

```
CONTEXT Grad_Courses[c# < 5000]
        SELECT Grad_Courses[c#, title]
DISPLAY
```

Query Transformations:

Initial:

Grad_Courses[c# < 5000].

Step 1: Rule 32 can be employed to accomplish semantic expansion. Notice that the initial query context contains an attribute restriction on c# so sub-rule b of Transformation Rule 2 must be applied. The resulting context is:

(Grad_Courses[c# < 5000]) • (Grad_Courses[c# >= 5000]).

Step 2: Transformation Rule 6, sub-rule c, is applied after step 1 to deduce that a logical contradiction exists between attribute restrictions in the context expression. The query result must be null, and thus the context expression simply reduces to:

NULL.

Rule 32 does not need to be user-defined, as it represents how the Grad_Courses sub-database is generated. From the deductive Rule 31, we know that every c# in the Grad_courses sub-database will have a c# greater than or equal to 5000 since this restriction is exactly the criteria for generating the sub-database. This case exemplifies the application of Transformation Rule 8.

In summary, this chapter has defined and demonstrated the use of transformation rules to accomplish semantic query optimization.

CHAPTER 6
COST ESTIMATION, FUTURE WORK, AND CONCLUSION

While Chapter 5 introduced transformation rules and demonstrated their use by case studies, it left the issue of gain accrued by transformations unanswered. Specifically, it never discussed what transformations would result in more profitable execution plans, how many transformations steps should be allowed for a given query, how many rules should be considered for a query, or in what order transformations should be performed. These are all issues of costing strategy and will be briefly discussed in this final chapter. Their introduction at this point is intended to be a guide for future research in this area. This chapter will also examine what future work must be accomplished to realize an implementation of semantic query optimization in OSAM*. Finally, a conclusion to this thesis will be given.

Cost Estimation

Profitability of semantic query optimization depends on many factors, not the least of which is the ability to estimate which plans will execute most efficiently. To make such an estimate, underlying data structures, data access paths, and the amount of data that is to be processed are some of the most

important considerations. For example, the existence of indexes, hash tables, sorted data files, an clustered data can greatly determine which plans offer the best execution strategy. Likewise, some of the plans may offer a very quick reduction in the amount of data being requested, and thus may be more beneficial in this aspect. Finally, if the identified context of the query is a small set of data in the first place, no optimization may be profitable since the overhead to perform the optimization may outweigh the cost to execute the original query.

These are exactly the types of concerns of conventional query optimization, except now their importance is even greater. No longer is it sufficient just to generate a multitude of plans, estimate which of the plans is most efficient, and then execute it. Such "passiveness" in the generation of plans would lead to far too many rules being considered for transformations (in the case of recursive rules, a never-ending list), far too many transformation steps, and the costing of far too many execution plans.

Costing elements must therefore be used to guide the transformation process as to limit the scope of its considerations and to direct the process along profitable avenues. Such guidance can be implemented by identifying which rules most often lead to the most efficient plans. For example, a particular semantic property of data may not be well known to users. As such, the rule representing this property generally leads to transformations that result in a null

answer and thus an efficient plan. Such a rule then should be given a high priority among those being considered for transformations. Likewise, a rule's consequent could represent a restriction that allows access to data through indexes or clusters. It too then should be deemed as highly profitable. In essence, the efficiency considerations that are typically identified in conventional query optimization must be used to identify which rules will result in transformations that are profitable and in what order transformations should be performed.

Another important consideration is semantic query optimization is the complexity of a query. In addition to estimating the value of a rule (in terms of the efficiencies it will introduce to query processing), the original complexity of the query must be estimated to determine the scope of the transformation process. In general, very complex queries can afford to search harder for efficient plans, for without such plans the execution time of the queries would be unacceptably long. It is worthwhile to pay for the increase in query compilation time to achieve a lower query execution time, especially if the query is complex and is evaluated frequently. Simple queries, on the other hand, can afford only a few transformations before the cost of query optimization becomes be higher than the cost of executing the query in its original form. Query complexity is vital in determining the scope of the optimization efforts.

As a final note on costing strategy, transformations concerning deductive rules demand special consideration (as alluded to in previous chapters). If a deductive rule is to be used in semantic expansion, as in Case Study 5 of Chapter 5, a couple efficiency restrictions must be maintained. First, the sub-database being derived must be materialized prior to query processing. Derivation during query execution is far too inefficient. Second, the derived sub-database must represent a significantly smaller subset of data than the amount of data the query initially would have been run against. These additional stipulations ensure deductive rule transformations in semantic query optimization are profitable.

Future Work

To implement semantic query optimization is OSAM*, the above costing strategy has to be realized. The following section suggests a methodology to realize the strategy. The entire costing strategy can be analyzed using a liability and benefit scenario. The initial query's context can be used to determine an "allowance" as to how much liability the query can afford based of the query's complexity. Highly complex queries will have higher allowances than simpler ones. Pertinent rules are then examined by their liability and benefit. The liability is the cost of transforming the query by that rule, while the benefit is the expected savings that result by using the rule. Only those rules whose benefit outweighs their liability should be considered for semantic query

optimization. The rules should also be ordered according to the difference of their benefit minus their liability, with the rules having the greatest difference being given the highest priorities. Each rule's liability then is considered as a deduction to the query's allowance. Rule's can continue to be considered for transformations as long as a query's allowance does not expire. Once the allowance is expired, the cost of the plans generated up until that point can be used to determine which has the most efficient execution strategy.

This methodology fulfills the essentials of the costing strategy. It limits the rules considered for transformations and orders them according to an estimated profitability. In doing so, it will also limit the number of transformation steps and the number of QEPs generated. The priority of rules also serves as a guide towards the most profitable transformations. This methodology mandates, however, that a query's complexity and a rule's benefit and liability be estimated. So far no mechanism has been developed in OSAM* to make such estimations. Also, the mapping of the data structures and access paths to rule's benefit has to be determined. Likewise, a rule's liability must be studied and determined from its transformations cost (i.e., the overhead it adds to query compilation).

Once the basic semantic query optimization is accomplished in OSAM*, there are several extensions that can be implemented. For example, database usage patterns may be used to determine new access strategies or to determine

which sub-databases defined in deductive rules should be materialized because of their frequent reference in queries. Likewise, the efficiency gained by using particular knowledge rules in semantic query optimization may be monitored to implement a more dynamic determination of a rule's benefit and liability. Another possibility for optimization in the OSAM* knowledge base management system is to monitor query histories so that queries with a high frequency of execution are allotted higher allowances for transformations (instead of solely considering query complexity for this purpose). Query histories may also be used to determine which queries are often executed together. Knowledge rules used for transformations of those queries executed together can be examined as a group to determine if any additional semantics may be deduced from logically grouping the rules together. This approach parallels the singularity of rules: one rule to a semantic property (i.e., Transformation Rule 6). Finally, the KBMS can track certain frequently queried data as to introduce state rules based on the values of that data. Such rules would not be integrity constraint rules, but simply rules indicating the current status of the database state. The maintenance cost for such a scheme must be considered..

Conclusion

In conclusion, this thesis has served several purposes. First, it was a review of the OSAM* knowledge base management system, including its S-diagram, query and rule specification languages, and association algebra. Next,

an overview of query optimization techniques has been presented. Particular attention was given to the conventional and semantic query optimization. Thirdly, a categorization scheme for rules was given and integrity constraint rules associated with OSAM* constructs were fully described. An emphasis was made that state integrity constraint rules are most beneficial to semantic query optimization. Finally, eight transformation rules, which make use of the rules of a knowledge base to optimize queries, have been introduced. The use of these rules for query optimization have been illustrated in a number of case studies. These case studies have shown that constraint and deductive rules of a knowledge base can be used by a query optimizer to generate meaningful and potentially profitable execution plans.

APPENDIX
LIST OF RULE EXAMPLES

Below is a compiled list of all the rule examples used in this thesis, numbered and ordered in the manner in which they were presented.

Rule 1:
IF Teacher * Section * Course
THEN Teacher-Course(Teacher[ss#,name],Course[c#,title]).

Rule 2:
IF Student[gpa > 3.5]
THEN Honor-Student(Student).

Rule 3:
IF Faculty[degree == "PhD"]
THEN title := professor ELSE title := "teacher".

Rule 4:
IF Student * Transcript * Course
THEN gpa := SUM (Grade by Student)
DIV COUNT (Course by Student).

Rule 5:
IF Faculty[degree == "PhD"] * Section * Course
THEN PhD-Faculty-Course (Faculty[ss#,name,degree,
title := "professor"], Course[c#,title]).

Rule 6:
IF PhD-Faculty-Course
THEN NOT_EXIST (PhD-Faculty-Course * Teaching-Assistant).

Rule 7:
IF Engineer THEN Employee * Engineer.

Rule 8:
IF Employee * Engineer THEN Employee ! SupportEmp.

Rule 9:
IF Employee * SupportEmp THEN Employee ! Engineer.

Rule 10:
NOT_EXIST (Employee * AND(Engineer, SupportEmp)).

Rule 11:
IF Part * Geometric THEN Part * Machining.

Rule 12:
IF Part * Machining THEN Part * Geometric.

Rule 13:
IF (Part * OR(Geometric, Machining))
THEN (Part * AND(Geometric, Machining)).

Rule 14:
IF Container THEN Container * OR(Device, Tray).

Rule 15:
NOT_EXIST (Container ! OR(Device, Tray)).

Rule 16:
IF Employee[Sex == "F"] THEN Employee * FemaleEmp.

Rule 17:
IF Employee[Sex == "M"] THEN Employee * MaleEmp.

Rule 18:
MAPPING (ClassX, X2, 1:1).

Rule 19:
IF Section THEN Section * Course.

Rule 20:
z3_Attr != NULL.

Rule 21:
IF Supplier THEN Supplier * Supply.

Rule 22:
IF Supplier * Supply * Part
THEN MAPPING (Supplier, Part, n : m).

Rule 23:
IF Supply THEN Supply * AND(Supplier, Part).

Rule 24:
IF Teacher ! Section THEN DELETE Teacher.

Rule 25:
IF Teacher THEN Teacher * Section.

Rule 26:
IF Teacher ! Section THEN NOT_EXIST Teacher.

Rule 27:
IF Employee THEN Employee * OR(Engineer, SupportEmp).

Rule 28:
IF Employee * Engineer THEN Employee.salary > $40k.

Rule 29:
IF Employee * SupportEmp THEN Employee.salary < $30...

Rule 30:
IF Employee THEN Employee * OR(MaleEmp, FemaleEmp).

Rule 31:
IF Course[c# >= 5000] THEN Grad_Courses[c#, title, college].

Rule 32:
IF Grad_Courses THEN Grad_Courses[c# >= 5000].

REFERENCES

[ALA89a] Alashqur, A.M., Su, S.Y.W., and Lam, H., "OQL -- An Object-oriented Query Language," Proceedings of the 5th International Conference on Very Large Databases, Amsterdam, The Netherlands, 1989, pp. 433-442.

[ALA89b] Alashqur, A.M., "Constraints in Object-oriented Databases," Technical Paper, Database Systems Research and Development Center, University of Florida, Gainesville, FL, July 1989.

[ALA89c] Alashqur, A.M., "Deductive Object-oriented Databases," Technical Paper, Database Systems Research and Development Center, University of Florida, Gainesville, FL, 1989.

[CHA85] Chakravarthy, U.S., "Semantic Query Optimization in Deductive Databases," Ph.D. Thesis, Department of Computer Science, University of Maryland, College Park, MD, August 1985.

[COD70] Codd, E.F., "A Relational Model of Data for Large Data Banks," Communications of the ACM, 13(6), June 1970.

[FRE87] Freytag, J.C., "A Rule-based View of Query Optimization," Proceedings of the ACM SIG Conference on the Management of Data, pp. 173-180.

[GIR78] Girshman, R., "The Simplification of Retrieval Requests Generated by Query-Answering Systems," Proceedings of the 4th International Conference on Very Large Data Bases, West Berlin, Germany, Sept 13-15, 1978, pp. 400-406.

[GOE87] Goetz, G., "Rule-based Query Optimization in Extensible Database Systems," PhD. Thesis, University of Wisconsin, Madison, WI, 1987.

[GOE89] Goetz, G. and Ward, K., "Dynamic Query Evaluation Plans," Proceedings of the 1989 ACM SIGMOD International Conference on the Management of Data, pp. 358-366.

[GUO89] Guo, M.S., Alashqur, A.M., Lam, H., and Su, S.Y.W., "An Association Algebra for Processing Object-oriented Databases," Technical Report, Database Systems Research and Development Center, University of Florida, Gainesville, FL, July 1989.

[HAM80] Hammer, M. and Zdonik, S.B. Jr., "Knowledge-Based Query Processing," Sixth International Conference on Very Large Databases, 1980, Montreal, pp. 137-147.

[JAR84a] Jarke, M. and Koch, J., "Query Optimization in Database Systems," ACM Computing Surveys, 16(2), June 1984, pp. 111-152.

[JAR84b] Jarke, M. and Vassiliou, Y., "An Optimizing Prolog Front-end to a Relational Query System," Proceedings of the 1984 ACM SIGMOD Conference, Boston, MA, June 1984, pp. 296-305.

[KIN81] King, J.J., "QUIST: A System for Semantic Query Optimization in Relational Databases," Proceedings of the 7th International Conference on Very Large Data Bases, Cannes, France, Sept 9-11, 1981, pp. 510-517.

[LAM89] Lam, H., Su, S.Y.W, and Alashqur, A.M., "Integrating the Concepts and Techniques of Semantic Modeling and the Object-Oriented Paradigm," to appear in IEEE COMPSAC, Orlando, FL, Sept. 1989.

[LOB88] Lobo, J. and Minker, J., "A Metaprogramming Approach to Semantically Optimize Queries in Deductive Databases," Proceedings of the Second International Conference on Expert Database Systems," Tysons Corner, VA, April 25-27, 1988, pp. 371-385.

[RAS88] Raschid, L. and Su, S.Y.W., "A Transaction Oriented Mechanism to Control Processing in a Knowledge Base System," Proceedings of the International Conference on Expert Database Systems, Tysons Corner, VA, April, 1988, pp. 353-373.

[SHE87] Shenoy, S.T., and Ozsoyoglu, Z.M., "A System for Semantic Query Optimization," Proceedings of the ACM SIG on the Management of Data, 1987, pp. 181-195.

[SIE88] Siegel, M.D., "Automatic Rule Derivation for Semantic Query Optimization," Proceedings of the Second International Conference on Expert Database Systems," Tysons Corner, VA, April 25-27, 1988, pp. 371-385.

[SU85] Su, S.Y.W., and Raschid, L., "Incorporating Knowledge Rules in a Semantic Data Model: An Approach to Integrated Knowledge Management*," Artificial Intelligence Applications Conference, Miami, FL, Dec 11-13, 1985, pp. 250-256.

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

[SU88a] Su, S.Y.W., and Siggelkow, J.A., "Semantic Diagram Representation for OSAM*", Technical Report, Database Systems Research and Development Center, University of Florida, Gainesville, FL, July 1988.

[SU88b] Su, S.Y.W., Database Computers: Principles, Architectures, and Techniques, McGraw Hill, New York, 1988, pp. 47-53.

[SU89] Su, S.Y.W., Krishnamurthy, V., and Lam, H., "An Object-oriented Semantic Association Model (OSAM*)," in Artificial Intelligence: Manufacturing Theory and Practice, S. Kumara, A.L. Soyster, and R.L. Kashyap, Institute of Industrial Engineers, Industrial Engineering and Management Press, Norcross, GA, 1989, pp. 463-494.

[XU83] Xu, G.D., "Search Control in Semantic Query Optimization," Technical Report #8309, Computer and Information Science Department, University of Massachusetts, Amherst, MA, 1983.

Capt Gleason was selected for attendance at the Air Force Institute of Technology's master's degree program in computer science, data processing for the fall of 1988. He has since been enrolled as a graduate student at the University of Florida.