INCORPORATING SEMANTIC INTEGRITY CONSTRAINTS IN A DATABASE SCHEMA

by

Heng-Li Yang

B. Sc., National Chiao Tung University, 1978
M. Commerce, National Cheng Chi University, 1980
M. Sc. (Computer Science), Pennsylvania State University, 1985

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

THE FACULTY OF COMMERCE AND BUSINESS ADMINISTRATION

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

August 1992

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Commerce and Business Administration

The University of British Columbia
Vancouver, Canada

Date August 6, 1992

Abstract

A database schema should consist of structures and semantic integrity constraints. Semantic integrity constraints (SICs) are invariant restrictions on the static states of the stored data and the state transitions caused by the primitive operations: insertion, deletion, or update. Traditionally, database design has been carried out on an ad hoc basis and focuses on structure and efficiency. Although the E-R model is the popular conceptual modelling tool, it contains few inherent SICs. Also, although the relational database model is the popular logical data model, a relational database in fourth or fifth normal form may still represent little of the data semantics. Most integrity checking is distributed to the application programs or transactions. This approach to enforcing integrity via the application software causes a number of problems.

Recently, a number of systems have been developed for assisting the database design process. However, only a few of those systems try to help a database designer incorporate SICs in a database schema. Furthermore, current SIC representation languages in the literature cannot be used to represent precisely the necessary features for specifying declarative and operational semantics of a SIC, and no modelling tool is available to incorporate SICs.

This research solves the above problems by presenting two models and one subsystem. The E-R-SIC model is a comprehensive modelling tool for helping a database designer incorporate SICs in a database schema. It is application domain-independent and suitable

for implementation as part of an automated database design system. The SIC Representation model is used to represent precisely these SICs. The *SIC elicitation subsystem* would verify these general SICs to a certain extent, decompose them into sub-SICs if necessary, and transform them into corresponding ones in the relational model.

A database designer using these two modelling tools can describe more data semantics than with the widely used relational model. The proposed *SIC elicitation subsystem* can provide more modelling assistance for him (her) than current automated database design systems.

Table of Contents

List of Figures

Acknowledgement

I wish to take this opportunity to express my sincere gratitude to all members of my dissertation committee, Professor Alvin Fowler, Dr. Robert C. Goldstein, Dr. Veda C. Storey, Dr. Yair Wand, and Dr. Carson Woo. In particular, I am especially indebted to my research supervisor, Dr. R. C. Goldstein, for his patience, support and countless hours of valuable discussions. I am also grateful to Dr. V. Storey because she suggested this great research topic and gave her opinions all the way. Many thanks go to Dr. Y. Wand for his critical and stimulating comments. I appreciate Dr. C. Woo for his comments on the dissertation organization and Professor Al. Fowler for his practical suggestions. In addition, I would like to thank my friend, Dr. Hsueh-Ming Hang, and all staff of the inter-library loan division of UBC Library, for their helping collect the related literature in the early stage of this research. Finally, I thank my parents for their love and support all the time.

Chapter 1

Introduction

1.1 Database Design

Database management systems (DBMSs) have been available for more than two decades. However, database design has also been recognized as a task with a high level of complexity ([Obretenov, et al., 1988]) and has often been referred to as an art rather than a science ([Holsapple, et al., 1982]). *Database design* is the process of modelling the information requirements of a real-world application and mapping them onto an underlying DBMS. Database design must go through the following phases: (1) information requirement elicitation, during which requirements for knowledge of a real-world application are determined; (2) conceptual database design, which produces a high-level representation of the requirements independent of the DBMS that will be used, e.g., the output might be expressed as an Entity-Relationship (E-R) model; (3) logical database design, which produces a logical schema that corresponds to the data model of the chosen DBMS, e.g., the output might be expressed as a relational data model; and (4) physical database design, which transforms the logical design into a form that is suitable for the given hardware and DBMS, and considers efficiencies of storage and processing.

Traditionally, database design has been carried out on an ad hoc basis ([Bouzeghoub et al., 1985], [Goldstein, 1985]). It has usually been performed by a human "database

design expert". The weaknesses of the traditional approach are that: (1) it is a difficult task and expert database designers are scarce; and (2) the design of a database is done by someone who is unfamiliar with the application domain, instead of end-users ([Storey and Goldstein, 1991]). Recently, a number of computerized systems have been developed for assisting the database design process ([Ram, 1989], [Storey and Goldstein, 1990]). Some of those can be classified as knowledge-based or expert systems; others are automation tools. Those systems are designed for assisting conceptual and/or logical design processes. Some even provide help for physical database design. Automation of database design process can help overcome the problems of the above traditional approach by codifying the database design methodology in a computer program.

However, some major problems remain. First, only a few of the above automated systems try to help a database designer incorporate semantic integrity constraints in a database schema ([Yang and Goldstein, 1989]). Many types of semantic integrity constraints have never been identified by any system. Furthermore, as observed by Troyer [1989, p. 423], "constraints often considered as first class citizens in the conceptual modelling seem to become pariahs during the transformation [from a conceptual schema to a relational schema]". Second, those automated systems usually only consider database states and static properties. They seldom consider the *behaviour* of a database, that is, the state transitions and dynamic properties. Some systems (e.g., E^2R [Kozaczynski and Lilien, 1988] and EXIS [Yasdi and Ziarko, 1987]) model the behavioral semantics by transaction modelling or event modelling. *Dynamic semantic integrity constraints*, which place restrictions on database transitions, have not really been treated as constraints on data.

1.2 Semantic Integrity Constraints

Semantic integrity is concerned with the logical meaning (i.e., the intension) of stored data and preserving the correctness of database contents even though users and application programs try to modify it incorrectly ([Fong and Kimbleton, 1980], [Fernandez, et al., 1981]). A database schema consists of structures and *semantic integrity constraints* (hereafter abbreviated as SICs) ([Tsichritzis and Lochovsky, 1982], [Frost, 1984]). The structure part tells us relatively little information other than the basic structure — what elementary items of data are grouped into what larger units; but the SIC part provides information about all allowable occurrences — current and future ([Morgenstern, et al., 1989]). These SICs express *data integrity semantics*, that is, the part of the meaning of stored data needed to determine correctness. They are invariant restrictions on the static states of the stored data and the state transitions caused by the primitive operations: insertion, deletion, or update. They express what is and is not allowed in the part of the universe that is represented by the stored data[1].

Traditional database design techniques focus on structure and efficiency. The relational database model is the popular logical data model with a good theoretical foundation. *Data dependency* (e.g., functional dependencies, and multivalued dependencies) theory has been well-formalized in the literature (e.g., [Delobel, 1978], [Ullman, 1982]). However, data dependencies only capture part of semantic integrity. A relational database in fourth or fifth normal form may still represent little of the *data semantics*, that is, the meaning of stored data. In addition, as observed by Kent [1979, p. 127], "The assumption tends to be that functional dependencies (if specified at all) have been used during the design phase of the database to insure that relations are in third normal form, and

[1]SICs also provide information that can be used to determine the most appropriate structure for the schema.

then discarded. They do not seem to be present at run time to explain the semantic structure of the data."

Limitation of SICs Although a database with embedded SICs would be more correct than the same one without SICs, the absolute correctness of the database is still not guaranteed. For instance, a user might incorrectly update the salary of an employee but, as long as it was within the range allowed by the SICs, it would still be accepted by the DBMS[2]. Also note that enforcement of a SIC is based on the assumption that all data already stored in the database are correct. If we know that some data were entered incorrectly, the related SICs may need to be turned off in order to correct these errors[3].

In addition, SICs are passive restrictions on data. By incorporating violation actions to alter other objects when a SIC is violated, one can make the database more active ([Morgenstern, 1983]). However, a SIC can only trigger some action when it is violated.

1.3 SICs, "Constraints" and Transactions

The reader should be cautious that in the literature the word "constraints" is sometimes used to include all abstract relationships between objects in an information system, e.g., the correctness of mapping to physical storage, and those preserving reliability, concurrent consistency, and security. The SICs discussed in this dissertation are a proper subset of

[2]Similarly, Thompson [1989, p. 95] points out that "a semantic database does not, and cannot, provide *meaning*, or *strong-semantics*, but it provides *contexts* within which it is possible for data to be meaningful". He names these contexts as *weak-semantics*. This dissertation does not adopt that term, but the reader should be cautious of the limitation.

[3]For instance, if a SIC states *"salary never decreases"*, an update from $12,000 to $10,000 will be rejected. However, it is possible that the $12,000 was entered incorrectly the first time. The SIC must be turned off in order to correct the input error.

these more general "constraints" (e.g., [Shepherd and Kerschberg, 1986]) "laws", or "sub-laws" ([Paulson, 1989], [Wand and Weber, 1988; 1989; 1990]). The following are some kinds of "constraints" that are not dealt with in this research.

Other Database Constraints In a database management system, there are at least four major aspects to the prevention of errors in a database environment: reliability, concurrent consistency, security, and semantic integrity ([Hammer and McLeod, 1975], [Eswarn and Chamberlin, 1975]). Reliability is concerned with errors due to the malfunctioning of system hardware or software. Concurrent consistency is the prevention of inconsistencies that may arise due to concurrent processing (in which multiple processes concurrently operate on shared data). Security deals with preventing users from accessing and manipulating the data in unauthorized ways. Although unauthorized updates to the database are sometimes said to violate the "integrity" of the system, this type of error is a security problem and is not considered in this research.

Non-database Constraints Database design is part of information system development, which is the process of modelling a portion of the real world and transforming it into an implemented artifact to deal with information processing functions in an organization. SICs are facts about the stored data ([Oren, 1985]) and are used to capture data integrity semantics ([Dampney, 1988]). However, data integrity semantics do not include all *information system semantics*, that is, knowledge represented in the information system. Some constraints are inherent to application programs or transactions rather than data ([Flemming and Halle, 1989, p. 140]).

Transaction Modelling Data-oriented system modelling has long been criticized for focusing only on static properties of an information system. There is ongoing research on how to add dynamics to data-oriented modelling. For example, the ACM/PCM (Active and Passive Component Modelling) uses SHM+ (the Extended Semantic Hierarchy Model) to include both structural and behavioral properties ([Brodie and Ridjanovic, 1984].) The popular conceptual modelling tool, the E-R model, has also been extended to model behavioral aspects of the real world by defining transactions or events (e.g., applying the *Petri Nets* technique [Sakai, 1983a], [Solvberg and Kung, 1986])[4]. A transaction, sometimes called an application-oriented operation (e.g., "hire") ([Casanova and Furtado, 1984]), consists of one or more database querying and/or altering primitive operations that must be treated as an atomic unit, and reflects changes and "happenings" in the real world. In transaction modelling, transaction specifications often include pre-conditions and post-conditions. The pre-conditions specify what must be true before the transaction can be applied. The post-conditions specify the actions to be taken and the test-conditions that should be true after the transaction.

Transaction-driven Constraints An information system is an artifact that relies on transactions to track changes in the real world. SICs place logical restrictions on stored data and are independent of any particular transaction. Transaction-driven constraints are inherent to transactions rather than data, and assure consistency between the information system and the real world. A transaction-driven constraint requires that when a change or "happening" occurs in the real world, a transaction must be performed

[4]Similarly, there are some information system development methods, e.g., the Z approach ([Spivey, 1988]) and VDM (Vienna Development Method) ([Jones, 1986]) to express formal specifications of static and dynamic aspects of information systems by modelling transactions or events.

to modify the database faithfully. For example, pre-conditions of a transaction may include some procedural or manual checks (e.g., issuing a message to ask whether there are signed documents or calls from customers). They are more prone to change as an organization evolves. They might include rules on some objects (e.g., virtual fields) in the information system that are not modelled as stored data in the database. They are likely to require a great deal of human checking since they may involve non-data objects (e.g., the above mentioned signed documents).

Note that because a transaction consists of primitive operations, it must also conform to all of the SICs on data. It might be inefficient to enforce SIC checking when a transaction is performed. It is possible to design an algorithm to transform SICs into the pre-conditions and post-conditions of transactions. For example, Lipeck [1986] describes some general rules for these transformations based on his temporal logic language for dynamic SICs. In fact, most of the usual pre- or post-conditions of transactions discussed in the database literature are transformed SICs rather than true transaction-driven constraints.

To illustrate, consider the following example. A clerk receives a telephone call from a customer to place an order. A transaction-driven constraint stipulates that a transaction "new-order" must be performed. It may stipulate that the transaction must be performed exactly according to the memo written by the clerk or the cassette recording of the call. In addition, it may stipulate that the transaction should be performed immediately by the clerk or in batch during the night by a computer operator. If the transaction is to be performed, a record of the order is to be "inserted" into the database. SICs would then check the attributes of an order, the existence of the customer in the database, etc. However, since SICs are intensional expressions, they do not, in general, restrict when an order must be put into the database or to whom the order should be shipped,

etc. In this example, in practice, it would also be impossible to use a SIC to increase automatically the customer's account balance by the total price of the order unless all past orders and payments are kept in the database to allow evaluation of an invariant formula among these objects[5]. Sending a bill to a customer may also involve a number of transaction-driven constraints in addition to SICs.

1.4 Enforce Semantic Integrity Constraints via Application Software

In traditional data modelling, database design is concerned with the structure of the data and most integrity checking is left to the application programs (procedures). Fernandez, et al. [1981, p. 109] identify the problems of relying on application programs for integrity checking as follows.

- Checking is likely to be incomplete because the application programmer may not be aware of the semantics of the complete database;
- Each application program must trust other programs that modify the database — one rogue program could corrupt the whole database;
- Code to enforce the same SICs occurs in a number of programs, wasting programming effort and risking inconsistencies;

[5]One should note that *statement-1* "the new customer balance is equal to its old balance plus total price of the order" is not an invariant assertion of a SIC for updating *Customer.Balance*. That statement is specific for the transaction, *New_Order*, to update the *Customer.Balance*, and does not apply to other transactions (e.g., a new payment by the customer or an update of the *Order.TotalPrice*), which also access these same data objects (i.e. *Customer.Balance* or *Order.TotalPrice*). The invariant formula in this example would be that *Customer.Balance* is equal to the sum of *Order.TotalPrice* minus the sum of *Payment.Amount*. However, this formula will normally be inefficient to check. For efficiency reasons, we may transform this formula into pre- or post-conditions on transactions. For example, *statement-1* would be a post-condition of the *New-Order* transaction.

- The criteria for integrity are buried in procedures and are therefore hard to understand and control;
- Primitive operations (update, insert, and delete) performed by users of high-level query languages cannot be controlled.

Fleming and Halle [1989, p. 140] state that traditional modelling methods typically leave the definition of SICs to application development rather than to database development. This application-driven method raises three dangers as outlined below:

- A user may define the SICs too narrowly, reflecting only the needs of the immediate application;
- Multiple users interested in different applications may have different perspectives on SICs. Thus they may define inconsistent or even conflicting SICs as part of the application specifications;
- Maintaining correctness and consistency across SICs implemented via multiple applications may be extremely difficult (or impossible) to accomplish as applications evolve and new applications arrive.

In summary, Fernandez et al., and Fleming and Halle identify the disadvantages of enforcing integrity via the application software as: **incomplete, inconsistent, redundant, incorrect, hard to understand and control,** and **difficult to maintain.** Using the database embedded SIC approach will overcome these disadvantages.

1.5 Embed Semantic Integrity Constraints in a Database

Database Approach Compared to traditional information processing before the development of database concepts, the database approach is claimed to have the following advantages [Goldstein, 1985, p. 8]:

- Controlling data duplication and inconsistency;
- Facilitating the sharing of data among applications;
- Assisting the coherent management of data as a basic organization resource;
- Increased programmer productivity;
- Increased applications' reliability;
- Enabling quick, economical response to ad hoc requests for information;
- Protecting data from damage or unauthorized access;
- Providing data independence.

It is not surprising to find that the disadvantages of enforcing integrity via the application software criticized by Fernandez et al., and Fleming and Halle are similar to the disadvantages to distributing stored data to separate files instead of a database. The full advantages of the "true database approach" are still not achieved if only the corporate-wide "data" are included in the database, but the corporate-wide SICs are distributed to separate application programs.

Behavioral Modelling As stated in the previous section, research has been ongoing to enhance data-oriented system modelling by incorporating some behavioral modelling methods to model the state transition and dynamic properties of an information system. Some researchers claim that transaction modelling is part of database design (e.g., [Brodie, 1986], [Brodie and Manola, 1989]). Some researchers even propose using transaction specifications to replace SIC specifications (e.g., [Abiteboul and Vianu, 1985], [Lipeck, 1986]). However, even if we agree that a "complete" database design includes transaction modelling, there would be some disadvantages of distributing SICs to separate transactions (e.g., redundancy and maintenance problems) that are similar to the case of distributing SICs to separate application programs. Both dynamic and static SICs are logical restrictions on data and should be embedded in a database.

In summary, this research assumes that transaction modelling is still valuable, but SIC specifications are fundamental to behavioral modelling. The position of this research is similar to the idea of having both specifications in the BASIS approach ([Leveson et al., 1983]), the idea of hierarchical levels of database specifications proposed by de Castilho, et al. [1982] and Casanova and Furtado [1984], and the hierarchical specification layers used in the CADDY design environment ([Hohenstein and Hülsmann, 1991]). That is, a "complete" database design will produce the two levels of specifications as follows.

1. First-level specifications form the database schema that consists of structures and SICs. These are data-driven and tend to be more fundamental (stable) since they are not affected by the addition or deletion of transactions;

2. Second-level specifications consist of the specifications of transactions. The pre-conditions and post-conditions provide an effective way of implementing SICs and may contain some transaction-driven constraints.

Knowledge-based Perspective Databases have been criticized for lacking abstract knowledge[6] ([Wong and Mylopoulos, 1977], [Bubenko, 1980]), and inference capabilities ([Wong and Mylopoulos, 1977], [Brodie, 1986]). Recently, the development of knowledge-based systems (KBS) provides useful insights for database research. Some researchers (e.g., Jarke and Vassiliou [1984], Missikoff and Wiederhold [1986], and Kennedy and Yeh [1990]) have recognized that a KBS can provide a DBMS with better semantic modelling, reasoning ability, improved user interface, etc. A special type of system, *expert database system*, has been proposed to integrate a knowledge-based system (expert system) with a database system ([Missikoff and Wiederhold, 1986]). The approach to embedding SICs in the database fits this new trend. SICs are abstract knowledge that can be used to provide deductive capabilities to a database, that is, make a DBMS appear more "intelligent". An obvious example is the case where SICs are used to reduce knowledge-based query evaluation costs ([Brodie, 1986]). For example, some queries can be answered using only the semantics expressed through SICs ([Chakravarthy, et al., 1987]).

1.6 The Research Questions, Objectives, Methodology and Scope

Recently, a number of semantic data models (e.g., [Hull and King, 1987], [Peckham and Maryanski, 1988]) have been proposed to overcome the weaknesses of traditional data models (i.e., the hierarchical, network, and relational models) in modelling the semantics of the real world. However, semantic integrity constraints have not received the attention they deserve. The relational model is still the most popular logical data model. Almost all research on semantic integrity constraints in the literature is confined to exploring

[6]Bubenko [1980] defines two kinds of knowledge: (a) *concrete knowledge* is seen as facts, statements concerning individual phenomena, entities or relationships in the model of the environment; (b) *abstract knowledge* denotes such information which augments our interpretation of concrete information and by which we can draw inferences, conclusions of other facts.

the efficient enforcement (checking) methods for a few kinds of SICs in a relational database or deductive database. No suitable language exists to represent the features of SICs precisely: certain or uncertain; for what data object; operation-independent or operation-dependent; conditional or unconditional; strong, soft or self-correcting; and static or dynamic[7].

Research Questions Based upon the above observation, this research addresses the following questions:

> *Is it possible that the features of SICs can be precisely represented by some language when using an E-R model for conceptual modelling and a relational model for logical modelling?*
>
> *Can we provide a model to incorporate the necessary SICs in a database schema during conceptual modelling?*
>
> *Can we help a database designer capture these SICs by providing automated tools?*

Research Objectives To answer the above research questions, the following specific objectives were established.

1. *To develop a precise* **SIC Representation model** *for specifying the components of a SIC.* Since current languages are not sufficient to represent the features of a

[7]These features are described in detail in Chapter 2. An operation-dependent SIC is the one that must hold for some object upon some operations, but not upon other operations. Violating a strong SIC would cause errors — reject the operation. Violating a soft SIC would only get a warning message.

SIC when the structure part of a schema is in the E-R model or in the relational model, this research will first present a SIC Representation model that is extended and modified from the model proposed by Fernandez et al. [1981], Date [1983], and Bertino and Apuzzo [1984].

2. *To develop a comprehensive modelling tool,* called **the E-R-SIC model**, *for incorporating the necessary SICs during conceptual modelling.* Since the traditional E-R model contains only a few inherent SICs, this research proposes a comprehensive model, which is an extended E-R model, for incorporating SICs in a database schema.

3. *To propose conceptually* **the SIC elicitation subsystem of an automated database design system** *for helping a database designer capture the necessary SICs.* The SIC Representation model and the E-R-SIC model are complementary. The captured SICs are represented in terms of the Representation model. The SIC Representation model is a language or representation tool to represent SICs. Its function is similar to the E-R diagram that we use to represent the structural part when using an E-R model. Both the models are application domain-independent and are suitable for implementation as part of an automated database design system. Some problems, for example, the procedure to query the database designer to identify SICs, verification of the captured SICs for consistency and nonredundancy, reformulation and decomposition of a general SIC into operation-dependent sub-SICs, and transformation of the SICs referencing entities and relationships into corresponding ones referencing relations, need to be solved in order to have a workable automated database design system. A SIC originally obtained from the database designer may be *general*, i.e., it may be relevant to several objects on various operations. Decomposing such a SIC is to rewrite it into several *sub-SICs*

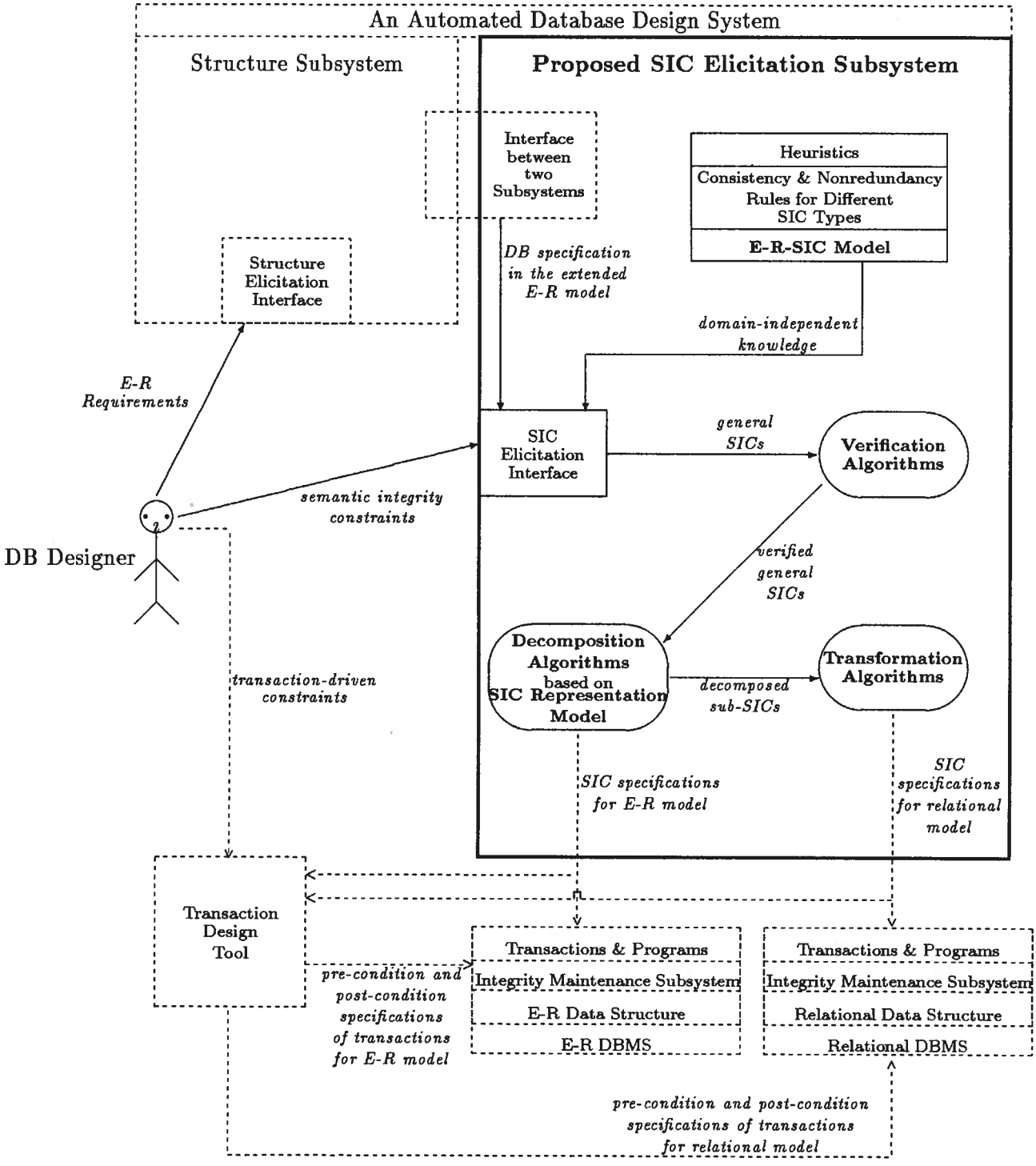
represented in terms of the SIC Representation model. Each of them is only relevant for one object on one access operation — that is, it is operation-dependent. The purpose of decomposition is not only to let the database designer know clearly what precise implications of a general SIC would be, but also to reformulate the original SIC into several formats that can be efficiently enforced later.

Research Methodology Methodologically, this research has two components. The first is *model building* to develop two models. The second is *algorithm designing* to develop algorithms to verify, decompose, and transform SICs.

Research Scope Figure 1.1 illustrates the scope of this research. This research is primarily concerned with semantic integrity constraints in the conceptual design phase and how they are translated into SICs in the logical design phase. In particular, it focuses on the Entity-Relationship model and its transformation into a relational model because of the popularity and wide-spread adoption of these as conceptual and logical data modelling tools, respectively.

A complete automated database design system would include a *structure subsystem* for constructing the structural entities, relationships, and relations, and a *SIC elicitation subsystem* for eliciting the SICs. The construction of the structure part of the schema is not the focus of this research although the *SIC elicitation subsystem* would need the constructed E-R diagram as input.

The proposed approach in this research is different from Codd's ([Codd, 1979]) although the purpose of capturing more meaning may be the same. Codd proposed a new data model RM/T to extend (indeed replace) the relational model. This research retains



Note that some general SICs needs to be first decomposed into precondition and predicate components before verifying them.

Legend: dash lines and boxes are beyond the scope of this research.

Figure 1.1: A Proposed Automated Database Design Subsystem for Eliciting SICs

the traditional relational model because of its popularity, but proposes to include the necessary SICs in addition to relation structures in a relational schema. The output of the proposed *SIC elicitation subsystem* would be SIC specifications, which are suitable for either the E-R model or the relational model. There should be an *integrity maintenance subsystem* in a traditional E-R DBMS[8] or relational DBMS. Some functional requirements (e.g., regarding SIC enforcement schedules and SIC inheritance, etc.) of this integrity maintenance subsystem are discussed in Chapter 4. However, in general, how this integrity maintenance subsystem would work is a future research topic beyond the scope of this dissertation. It is also assumed that the DBMS would support inheritance mechanisms.

This research does not address transaction modelling. The database designer may later input transaction-driven constraints and SIC specifications to a transaction design tool to produce the pre-conditions and post-conditions of transactions, which would be performed in an E-R DBMS or relational DBMS.

This research neither empirically tests the "usefulness" of using database embedded SICs versus the traditional enforcing integrity via the application software nor tests the "usefulness" of using the proposed automated database design system. The empirical research is a future research topic.

1.7 Contributions

The contributions of this research are both theoretical and practical.

The theoretical contributions include the following:

[8]It is assumed that there are some (probably experimental) DBMSs to define and manipulate data objects directly in the E-R model.

1. This research provides a model to represent precisely the features of a SIC.

2. This research also develops a model to incorporate the necessary SICs in a database schema. The approach is to model dynamic as well as static SICs in the database rather than in transactions or programs. The gap between traditional database modelling and application programming (transaction modelling) will be bridged by the two models presented in this research.

3. The work on reformulating and decomposing a general SIC into sub-SICs, and transforming them from an E-R schema into a relational schema may be interesting to the computer science discipline because the current literature has explored these problems for only a few kinds of SICs.

On the practical side, the contributions of this research include the following:

1. The proposed automated database design system would help a database designer not only design the database structure but also include the necessary SICs.

2. This research provides a foundation for overcoming the well-known problem of representing data integrity semantics in current relational database systems. The resulting database would have the advantages of embedded SICs, e.g., greater consistency, deductive capabilities, etc. The SIC representation would facilitate the efficient enforcement.

3. A database designer would find that modelling data integrity semantics becomes his (her) responsibility and right. Important business rules and even heuristics can enter the database schema in an early database design phase — conceptual modelling. Application programmers could focus on information system semantics other

than data semantics, and need not worry about SIC checking in their individual programs.

4. This research provides a starting point for future empirical research to test the usefulness of using database-embedded SICs versus the traditional approach of enforcing integrity via application software.

1.8 The Dissertation Outline

This chapter has given the definition of SICs, described the motivations, methodology, scope and contributions of this research. Chapter 2 briefly reviews work on SICs in the literature. Chapter 3 introduces the SIC Representation model for representing SICs uniformly and precisely. Chapter 4 describes how the SIC Representation model can be applied. Chapter 5 proposes the E-R-SIC model for incorporating SICs in a database schema. Chapter 6 gives an example of the use of the E-R-SIC model and discusses related issues and usefulness. Chapter 7 conceptually proposes a *SIC elicitation subsystem* to help the database designer use the E-R-SIC model and the SIC Representation model. Finally, Chapter 8 offers conclusions and describes how future researchers can extend this dissertation. A series of appendices is attached to provide related materials in detail and give some examples.

Chapter 2

Review of Previous Work

As briefly mentioned in Chapter 1, an automated database design system would first elicit general SICs from a database designer, then verify and reformulate (decompose if necessary) them in some representation language for the database structural schema represented in the E-R model, and finally transform them into corresponding ones in the relational model. This chapter reviews previous work on related aspects of producing SICs.

The literature on SICs is rich. Most research concentrates on classifying and efficiently enforcing SICs rather than capturing and incorporating them in a database schema. In addition, most research discusses SICs in the context of relational or deductive databases rather than the databases in the conceptual level of E-R schema. Therefore, transformation from the SIC representations in the E-R model into corresponding ones in the relational model has not been explicitly discussed in the literature although some existing automated database design systems may perform it for a few types of SICs that they identify. Section 2.1 first reviews the different ways to classify SICs because the classification can help understand the important SIC features so that we can incorporate, represent and enforce them properly. Section 2.2 reviews previous research on languages for representing SICs. Section 2.3 reviews how verification of SICs has been dealt with in the literature. Section 2.4 reviews previous research on reformulating SICs. Finally,

Section 2.5 reviews how SICs are handled by existing automated database design systems.

2.1 SIC Classification

In the literature, there are a number of different ways to classify SICs.

2.1.1 Classification Based on Explicitness

One way to classify SICs is based on their explicitness from the perspective of the data model in use. SICs can be inherent, explicit or implicit ([Tsichritzis and Lochovsky, 1982], [Brodie, 1983], [Brodie, 1984], [Davis and Bonnell, 1989]). *Inherent constraints* are integral parts of the structure of a data model (e.g., record relationships in the hierarchical data model are structured as trees; tuples in the relational data model are not duplicated and the ordering of tuples is not important). *Explicit constraints* are defined explicitly by some specification mechanism; for example, a database designer might explicitly specify that the salary of any employee must be less than $1,000,000. Finally, *implicit constraints* are logical consequences derived from inherent or explicit constraints; for example, the transitive closure of functional dependencies can be deduced from a subset of functional dependencies.

Discussion A data model with rich inherent SICs would relieve the database designer from specifying many SICs explicitly. From this perspective, neither the E-R model nor the relational model is a good modelling tool for designing a database since both models, especially the relational model, contain very few inherent SICs. However, they are widely used for other reasons. It is one of the motivations of this research to help

the database designer identify and precisely specify explicit SICs. A database designer and an automated database design system need to know the inherent SICs of the data model (e.g., the E-R model) that they use. Otherwise, it is likely that these inherent SICs would be lost when the structural schema of the conceptual model is transformed into the logical model (e.g., the relational model). Both inherent and explicit SICs should be represented properly at the conceptual design phase and transformed into a logical schema. Given a set of inherent and explicit SICs, the database designer should also be aware of the derived consequences, i.e., implicit SICs, to verify its consistency and non-redundancy.

2.1.2 Classification Based on Applied Objects

Classification of SICs based on applied objects is the most common classification and is concerned with the data objects to which a SIC applies ([Eswarn and Chamberlin, 1975], [Fong and Kimbleton, 1980], [Fernandez et al., 1981], [Tsichritzis and Lochovsky, 1982], [Date, 1983], [Weber, et al., 1983], [Simon and Valduriez, 1984]). There are three categories of these SICs:

1. *Strong data type constraints*: Strong data type constraints (also called *domain constraints*) are applied to a single data item, for example, a field or an attribute. They include the following:

   (a) *Value constraints* specify the range of acceptable values of a data item (e.g., within some numerical bounds or an enumerated set) and whether a null value is allowed.

(b) *Nonvolatility constraints* declare whether a data item value can be changed ([Kozaczynski and Lilien, 1988]).

(c) *Extended format constraints* permit specifications of data type, length, and format (mask) pattern;

(d) *Legal operation constraints* limit the operations that can be performed on a given domain ([Eswarn and Chamberlin, 1975], [Fong and Kimbleton, 1980]); for example, two dates cannot be multiplied.

2. *Record (tuple) constraints*: *Record constraints* apply to an individual record (an occurrence in terms of the E-R model). For example, in each payroll record, *Gross_Salary* should be greater than *Deductions*. One interesting kind of record constraint is the case where the value of one field in a record is conditional on the values of other fields ([Benci et al., 1976]). For example, *Salary is equal to Base_Salary plus Bonus*. In this example, regardless of whether the value of *Salary* is entered by a user or computed by the system itself, as long as this value is explicitly stored, the integrity constraint must hold. If the conditional attribute is not explicitly stored, it is a virtual invariant, derived item because one data item is defined as a function of another ([Etzion, 1989]).

3. *Set constraints*: *Set constraints* apply to a set of records (occurrences). They may be based on built-in aggregate functions (e.g., average, minimum, count). Therefore, they are sometimes called *aggregate constraints* ([Tsichritzis and Lochovsky, 1982]) or *set function constraints* ([Dogac et al., 1985]). They may be based on comparison (e.g., exclusion or inclusion) of one set to another. For example, *the set of managers must be contained in the set of employees*. These records (occurrences) may belong to the same relation (entity) type or different relation (entity) types. From this

observation, some researchers (e.g., [Fong and Kimbleton, 1980]) further classify them into relation constraints and multi-relation constraints.

Discussion This categorization reminds us that all objects in a database — attributes, entity and relationship occurrences and types in the E-R model, columns, relation tuples and types in the relational model — may have related SICs that need to be identified and represented. However, there are two controversial points.

- Should we capture legal operation constraints? Note that this kind of "constraint" borrows the notion of abstract data types ([Goldstein, 1985]), and the "operations" are basic arithmetic or string operations that may be performed on value domains, not the database primitive manipulation operations. This research will not consider these "constraints" because they do not conform to the SIC definition. This kind of "constraint" is relevant only if the data object is manipulated by some restricted "operations" (e.g., arithmetic or date operations). It is not a restriction on static database states or state transitions.

- Should we capture extended format constraints? One may argue that the specification of data type, length, and format of an attribute is a syntactic rather than a semantic issue. However, this research includes extended format constraints as part of SICs for the following two reasons. (1) The semantics of an attribute are based on the syntax we agree to use. For example, the meaning of a salary range from $1,000 to $50,000 in integers is different from that of the same range of real numbers. (2) SICs in this research are attached to the stored data rather than real world objects. A database designer would specify a SIC in terms of the interests of the application and include the data in the format that is meaningful to it.

2.1.3 Classification Based on Operation Type

SICs that are concerned with database operations can be classified as operation-dependent or operation-independent constraints ([Eswarn and Chamberlin, 1975], [Fernandez et al., 1981], [Weber, et al., 1983]).

- *operation-independent:* A constraint is operation-independent if it **must hold** for some object on all access operations, although for efficiency reason it **may be enforced** only on some operations.
- *operation-dependent:* A constraint is operation-dependent if it **must hold** for some object on some access operations, but **not** on other access operations.

Discussion We only need to consider three kinds of operation types: insertion, deletion and update because a retrieval (query) operation does not change any data. An update operation is on an attribute of an entity, relationship occurrence, or relation tuple, but not on the whole entity, relationship occurrence, or relation tuple. It is also the only applicable operation on an attribute. Therefore, if a SIC is relevant to an attribute, it is operation-dependent. For an entity, relationship occurrence, or relation tuple, there are two possible operations — insertion and deletion. It appears that in the real world if a SIC applies to an entity, relationship occurrence, or relation tuple, it is likely to be operation-independent. Operation-dependent constraints (e.g., "*a project can be deleted only if its budget is equal to zero*") seem to be relatively less common. This may explain why previous research pays only little attention to them. However, the possible relevance to specific operations creates a special requirement on the SIC representation language.

2.1.4 Classification Based on Precondition

According to Fernandez et al. [1981], SICs based on the precondition for enforcement can be classified as either conditional or unconditional. *Conditional constraints* are enforced only when certain preconditions are met. For example, the salary of an employee who has worked for less than three years must not be over $30,000. *Unconditional constraints* are always enforced.

Discussion Whether a SIC is conditional or unconditional is also relative to the object concerned. It seems that the research of Fernandez et al. [1981] is the only work that classifies SICs in this way. However, this categorization perspective places another requirement on the SIC representation language — the context under which the SIC is applicable should precisely be represented.

2.1.5 Classification Based on Certainties of SICs

SICs can be classified as certain or uncertain ([Oren, 1985], [Morgenstern et al., 1989]). A *certain constraint* specifies some fact about the data semantics that is assumed to be absolutely true (e.g., *"the height of a person is greater than zero"* [Oren, 1985]). An *uncertain constraint* is one that is generally true, but there is a slight probability that it can be violated (e.g., *"the weight of a person is less than 200 kgs"*). Certain and uncertain constraints lie on a continuum. Wiederhold [Morgenstern et al., 1989] identifies four levels with respect to the degree of certainty and absoluteness of constraints:

1. absolute truths that arise in the physical world and in the database; for example, *an employee has a birth date.*

2. rigid situational rules that do not change often; for example, *each employee is assigned to one department.*

3. business rules that may change often; for example, *a manager's salary is greater than the salary of his or her subordinates.*

4. heuristics; for example, *employees are usually assigned to projects that match their specialities.*

A more sophisticated proposal such as defining and maintaining "measure(s) of accuracy" may be possible ([Eswarn and Chamberlin, 1975]). Violation of a certain constraint may be rejected as an error, whereas violation of an uncertain constraint may only cause a diagnostic message. Therefore, uncertain constraints are sometimes called *soft constraints* ([Eswarn and Chamberlin, 1975]).

Discussion This classification is another perspective that has not drawn much attention. Since it is likely that an organization might have a number of uncertain but useful constraints, there should be some way to represent them.

Note that the four levels of certainty mentioned by Wiederhold ([Morgenstern et al. 1989]) are in fact classified by two kinds of "uncertainty" — **exception and permanence**. "Heuristics" may have exceptions. "Absolute truths", "rigid situational rules", and "business rules" have no exceptions. If there is any exception, the "rules" should be modified to accommodate the exception and the "modified rules" have no exceptions. These levels differ on their permanence — the "uncertainty" of how often they may be changed.

Should SIC specifications include both "uncertainty" of exception and "uncertainty"

of permanence? The "permanence" information may be useful for SIC management during database usage. However, the organizational environment is turbulent and constantly changing. The permanence of a rule is not easily decided in advance when a database designer designs a database. Therefore, this research does not include this kind of information.

2.1.6 Classification Based on Violation Action

SICs based on the violation action, i.e., what would happen if the SIC is to be violated, can be classified as *strong*, *soft*, or *self-correcting* ([Weber et al., 1983]). If a strong constraint is violated, the operation is rejected and the user receives an error message. If a soft constraint is violated, the user only receives a warning. If a self-correcting constraint is violated, its error correcting action is executed.

Discussion In the published literature, [Weber et al., 1983] is the only research that classifies SICs according to their violation actions although there is other research that discusses alternative violation actions. Violation actions are classified into the three categories given below, which are synthesized from previous research ([Hammer and McLeod, 1975; 1976], [Hammer and McLeod, 1976], [Casanova and Tucherman, 1988] and [Fleming and Halle, 1989]).

1. *Reject* — reject the requested database operation, signalling an error.

2. *Warning* — allow the requested database operation, but issue a warning.

3. *Corrective Action* — perform a corrective action; that is, an auxiliary procedure known as a triggered action ([Fernandez et al., 1981]). Usually, it causes the DBMS

to insert, delete, or update other objects.

For referential integrity constraints, e.g., $E.A \subseteq F.B$, where $E.A$ is a foreign key of a relation E and $F.B$ is the primary key of another relation F, the corrective action can be further classified as:

(a) *Propagate* — for example, insert a referenced tuple or delete the referencing tuples.

(b) *Nullify* — for example, set the referencing attributes in the referencing tuple to null.

(c) *Default* — for example, set the referencing attributes in the referencing tuple to predefined default values.

(d) *Others* — for example, triggers or other procedures that are domain specific.

Note that *"nullify"* and *"default"* are special cases of *"propagate"*.

Traditionally, a SIC is strong by default. However, this classification is important to remind a database designer that it is not necessary that if a SIC is violated, the operation is just rejected. There are other options that can be specified at database design time.

2.1.7 Classification Based on Enforcement Schedule

SICs based on the enforcement time, i.e., when the SIC is enforced, can be classified as immediate or deferred (e.g., [Eswarn and Chamberlin, 1975], [Fong and Kimbleton, 1980], [Fernandez et al., 1981], [Date, 1983], [Weber, et al., 1983]). *Immediate constraints* are enforced immediately after each database operation. *Deferred constraints* are not enforced until the end of a transaction. It may also be possible to permit the user to

switch integrity checking ON or OFF — that is, to have user-invocable constraints ([Fong and Kimbleton, 1980], [Bertino and Apuzzo, 1984], [Weber, et al., 1983]).

Discussion A number of researchers discuss this categorization. However, different specification levels should not be mixed up. This categorization is useful when considering the enforcement of SICs, or designing transaction specifications. Fernandez et al. [1981] allow the precondition component of their original model to specify whether the SIC is to be applied immediately or deferred to the end of a transaction or to a periodic audit. However, Bertino and Apuzzo [1984] treat the enforcement schedule separately as decided by an integrity maintenance subsystem because they believe that the enforcement schedule of a SIC depends upon the transactions or programs that are executing. Their example may be helpful to understand their arguments:

> A SIC: *"each employee working on project P125 must earn less than $3,000."*
> Suppose that now the database contains an employee entity occurrence who is in department D55, and works on project P125.
> Consider a transaction: *first update the salary of all employees in D55 to $3,500; then re-assign the project of all employees in D55 to P200.*
> This transaction should be correct.

Based upon the above observation, Bertino and Apuzzo [1984] propose a criterion:

> Basically, an integrity constraint is enforced at the end of the transaction, if attributes present in the constraint are modified by more than one update statement

> in the transaction. Otherwise, the constraint is enforced after each tuple-update, if it is in class C1 [i.e., tuple constraints], or after each update-request, if it is in class C2 [i.e., relation constraints] or C3 [i.e., multi-relation constraints].

Furthermore, the enforcement schedule can be more sophisticated and more efficient. Lafue [1982] proposes that constraint checking can be delayed until the dependent instances become of interest[9]. That strategy is called by Morgenstern [1986] "propagation when used." Between immediate propagation and "propagation when used" is an intermediate strategy that has been referred to by Morgenstern [1986] as opportunistic propagation, both in the sense of doing the work of constraint propagation when the computer is idle, and in the sense of using priority ranking of the constraints to select the order in which they should be considered for propagation.

Thus, because the enforcement schedule is closely related to transaction modelling and enforcement implementation efficiency strategies, it should not be included in SIC specifications. Neither should be the option to switch integrity checking ON or OFF.

2.1.8 Classification Based on Dynamics

SICs based on their dynamics can be classified as static or dynamic ([Eswarn and Chamberlin, 1975], [Bracchi et al., 1979], [Fong and Kimbleton, 1980], [Fernandez et al., 1981], [Date, 1983], [Bertino and Apuzzo, 1984], [Brady and Dampney, 1984], [Heuser and Richter, 1986]). *Static constraints* specify correct database states. *Transitional (dynamic) constraints* characterize valid state transitions, i.e., are concerned with "admissibility" of

[9] A dependent variable is a variable that can be operated on by the constraint, i.e., by a violation action.

a database state sequence.

There are two major kinds of transitional constraints mentioned in the literature ([Fong and Kimbleton, 1980]): (1) *old/new transitional constraints* that restrict an update of an attribute during which its "old" value is to be changed to a "new" value (e.g., *"new salary must be greater than old salary"*); (2) *nonexistence/existence transitional constraints* that restrict either a nonexistence to existence transition or an existence to nonexistence transition (e.g., *"only if the account balance is zero, can the account be deleted"*).

Some dynamic constraints (e.g., [de Castilho et al., 1982], [Casanova and Furtado, 1984], [Ehrich et al., 1984], [Kung, 1984] and [Lipeck, 1986]), which are often neglected by researchers, include:

- **constraints on a sequence of operations:** some operations must happen in a specific sequence or at the same time. For example, *"ownership of a car must be passed from a manufacturer to a dealer first, before it may be passed to a purchaser"*.
- **constraints involving time explicitly:**
    1. SICs having some explicit time restriction or time-triggering condition. For example, we may have *"an employee cannot receive a salary raise during his (her) first 6 months in the company"*, or *"at 0:00 on 1/1/1993, increase the salary of each employee by $1,000."*
    2. Time-triggered or restricted SICs depending on past generations of data and, thus reference historical data at some specific time point. For example, we may have *"the price of any product at any time cannot be more than 5% higher than its price one year ago."*

Discussion Both static and dynamic SICs are important to preserve the logical meaning of stored data. A database designer should capture the necessary dynamic SICs and have some language to represent them. Constraints on a sequence of operations are not easily captured and represented as SICs. This may explain why researchers often either neglect them or model them by transactions or events. However, they are the consequences of enforcing some SICs if specified properly.

Note that an operation-dependent SIC is also a dynamic SIC. Old/new transitional constraints and nonexistence/existence transitional constraints are important. However, one should not define them too narrowly. That is, old/new transitional constraints are special cases of the update transitional constraints, which do not necessarily involve the old/new comparison. Nonexistence/existence transitional constraints are also special cases of the deletion/insertion transitional constraints, which may involve more than one object.

In order to capture and represent SICs having explicit time restriction, in a real time environment, we would need a special system variable — *Current_time* — to register the current clock time[10], and explicit time-valued attributes in the entity or relationship. In a non-real time environment, those become ordinary data-driven constraints involving some time-valued attributes. If we wish to capture and represent SICs using historical data in general, we would need time-stamped generations of data. This research does not explore the last kind of SIC since a database keeping time-stamped generations of data is both unusual and expensive to implement.

[10]It is assumed that a DBMS has access to a clock that registers both current date and time. *Current_time* can be thought of as an attribute of a special system entity type. Because constraints may be affected by the normal advance of time, we assume that the operating system can be instructed to signal the DBMS integrity maintenance subsystem when a pre-specified time is reached.

2.1.9 Summary of Comments on SIC Classification

In the literature, researchers describe different SIC categorization schemes for diverse purposes — discussing data models, incorporating, representing, and enforcing SICs. For incorporating and representing SICs adequately, the most important categorization schemes are: (1) certain or uncertain, (2) the classification on applied objects, (3) operation-independent or operation-dependent, (4) unconditional or conditional, (5) strong, soft or self-correcting, and (6) static or dynamic . These categorizations are too rudimentary to serve as a modelling tool for incorporating SICs. However, they are very important for precisely representing SICs since they provide a feature listing of a SIC. Without any explicit description, one could only assume the given SIC to be certain, related to all objects mentioned in that SIC, operation-independent, unconditional, and strong.

2.2 SIC Representation

SQL is a generally accepted relational database language. However, only very few SICs are mentioned in the ISO SQL standard. The implied violation action of a SIC is to set SQLCODE negative. That is, it causes an error. Other SIC features, such as whether the SIC is operation-independent or operation-dependent, unconditional or conditional, certain or uncertain, are not specified.

The SQL standard has two levels and one addendum [van der Lans, 1989]. The specified SICs are as below:

1. **SQL Level 1 Standard:** It only specifies data types and length of data items. NOT NULL must be specified in every column definition in a CREATE TABLE

statement.

2. **SQL Level 2 Standard:** In addition to data types, it allows the designer to specify whether a data item is UNIQUE and whether it can be NULL.

3. **SQL with Addendum:**

   - In addition to the above, it allows specification of a value range by using the CHECK specification for a data item;
   - By using the CHECK specification separately (e.g., CHECK (YEAR-OF-BIRTH < YEAR-JOINED)), tuple constraints can be specified;
   - Few set constraints have been included:
     - Referential constraint is provided by using FOREIGN KEY (column list) REFERENCES (table name);
     - UNIQUE or PRIMARY KEY specification

A number of researchers propose alternative languages to represent more SICs. Some are variants of first order logic (FOL) languages, for example, the many-sorted first-order predicate calculus applied by Furtado et al. [1981]; the *constraint equation* proposed by Morgenstern [1983; 1984a; 1984b; 1986]; the *equation statements* suggested by Cosmadakis and Kanellakis [1985]; the first order formulas used by Reiter [1984; 1988], Henschen et al. [1984], and Urban and Delcambre [1989]. These FOL family languages are purely declarative. The operations to be checked are treated as an implementation problem. No violation action is specified. The constraints are assumed to be certain. Furthermore, these FOL languages cannot represent some kinds of SICs, e.g., operation-dependent or dynamic SICs. Other researchers (e.g., [de Castilho et al., 1982], [Casanova and Furtado, 1984], [Ehrich et al., 1984], [Kung, 1984] and [Lipeck, 1986]) propose an

extension of FOL — temporal logic that may include explicit "state" or "time" parameters. They invent some special temporal quantification or a list of modalities (e.g., *always, until, heretofore, sometime*) to model dynamic SICs. One potential problem of these temporal logic languages is that they may not be easily understood and implemented.

Others just extend the original SQL proposals, but do not follow the SQL standard. For example, Hammer and McLeod [1975] state that the syntax of their language is "rather similar" to SEQUEL. Bertino and Apuzzo [1984] propose a 5-component model to specify SICs and state that their language is a "simple extension" to SQL. Date [1983] applies a language of his own — very loosely based on the PL/I version of UDL (Unified Database Language) and describes [1987] some proposed extensions of the base SQL standard. These SQL extensions are more precise and powerful for representing SIC features than the SQL standard. Among these, the language model proposed by Bertino and Apuzzo [1984] and the similar one by [Fernandez et al., 1981][11] may be capable of representing the SIC features mentioned above except for certainties. However, neither work carefully elaborates what would be in each component[12]. In addition, SICs represented in their original model may not be efficiently enforced if we compare the representation to Date's UDL [Date, 1983] that uses the idea of a cursor to facilitate SIC enforcement on occurrences.

The certainty feature of SICs has not been properly represented. The term *fuzzy integrity constraints* has appeared in the *fuzzy database* literature (e.g., [Zvieli and Chen,

[11] However, [Fernandez, et al., 1981] does not describe what is really their language — first order logic or SQL.

[12] For example, The model proposed by Fernandez, et al., [1981] allows to specify the enforcement schedule as a part of precondition of a SIC. The model proposed by Bertino and Apuzzo [1984] allows the options to switch a SIC ON/OFF. However, as discussed in the previous section, these should not be in a SIC specification.

1986], [Raju and Majumdar, 1988]). However, its meaning may have not been fully explored. Raju and Majumdar [1988] classify fuzzy relations in relational databases into two categories. A type-1 fuzzy relation captures the impreciseness in the association among entities, e.g., the certainty of John liking the course AI is 80%. A type-2 fuzzy relation produces further fuzzy semantics by allowing the domain of an attribute to be a set of fuzzy sets, e.g., *"allowing salary of John to be in the range $40,000 to $50,000 and that of Mary to be a fuzzy set, low"*. However, their fuzzy integrity constraints only attach some fuzzy modifiers, e.g., "very", "more or less" to an assertion by choosing some "fuzzy resemblance relation", for example, "*for any job, employees having approximately equal experience should have approximately equal salary*" or "*any items having approximately equal order-date should have more or less equal delivery date*". This kind of SIC does not capture the "impreciseness" of a SIC itself. For example, "*any items having approximately equal order-date should have more or less equal delivery date*" is a SIC that is true with certainty 75%. RESTRICT is another language to consider the certainties of SICs ([Oren, 1985]). However, it simply uses a special operator "?" to represent an uncertain assertion, e.g., "SALARY(PERSON) <? 150000". This approach has only two levels of certainty — certain and uncertain.

In addition, all of the languages mentioned above represent SICs in the relational model, not in the E-R model. It is desirable to have a uniform language to represent all SICs in the E-R model and in the relational model. It would be easier for system analysts, database designers, and programmers to learn, comprehend and communicate with each other if there is a uniform language for both the conceptual modelling and logical modelling phases.

2.3 SIC Verification

It is suggested ([Bracchi et al., 1979] and [Morgenstern et al., 1989]) that it is necessary to verify a set of SICs based on some criteria: completeness, correctness, consistency, nonredundancy, no unexpected implicit constraints, and insensitivity to order.

In the database literature, no substantial results have been published that assure that a set of SICs is complete and correct, which would need both application-domain and general world knowledge.

If the SIC representation is not precise enough to capture the restriction intention, the enforcement outcome of a set of SICs may depend upon the order in which particular constraints are executed. Other remaining criteria — consistency, nonredundancy and no unexpected implicit constraints — are closely related although their verification difficulty may be increasing. Redundancy occurs in a set of constraints if some constraints subsume other constraints. It would be an issue only if we are concerned with efficient enforcement of a set of constraints. To assure that a set of SICs has no unexpected implicit constraints would require a database designer to understand well the closure of the set, i.e., all those consequences derived from the specified SICs. It requires that the database designer make judgements as to whether an implicit constraint is unwanted. The fundamental and most important issue is the consistency problem. Constraints are consistent if there exists a database state or a state transition that is allowable with regard to all of the restrictions. Unfortunately, the consistency problem has been described (e.g., [Meersman, 1988]) as a difficult one. SICs are either tacitly assumed to be consistent, or only some of them are verified for consistency. For example, Kung [1984] [1985] presents a tableaux method to check the consistency of restricted first-order SICs. Bry and Manthey [1986] describe two basic approaches to extending refutation methods into procedures to check

consistency of closed and function-free first-order SICs. Brodie [1978] mainly relies on an actual database to verify the consistency of some static SICs. Lenzerini and Nobili [1987] have contributed much to the consistency problem for cardinality constraints. In summary, past researchers have only tried to verify very few types of SICs for consistency. Furthermore, the redundancy problem of SICs has never been directly addressed in the literature. Nor has the issue of unexpected implicit constraints been explored for arbitrary SICs.

The major reason for this is that it has been known (e.g., [Nillson, 1980]) that even the "logical implication" of first order logic predicates is "undecidable" or only "semi-decidable"[13]. In this research, in order to cover more types of SICs, expressions in the precondition and predicate of a SIC could include not only first order, but also higher order logic. The full verification of consistency and non-redundancy of all SICs relates to the fundamental issue of computer science and mathematics — constructing a Turing machine to decide whether an arbitrary language is acceptable. Such an issue would be NP-complete in nature (i.e., putatively having exponential time complexity)[14].

Based on current techniques, we cannot verify a set of arbitrary SICs for consistency and nonredundancy. However, it is possible to classify SICs into several different types and analyze at least some of them for consistency and non-redundancy. Some researchers have tried it, but only for very few types of SICs and produced relatively rudimentary results (e.g., Troyer [1989] verified five types of SICs for consistency in the Binary Relationship model, Furtado et al., [1988] verified two types of SICs for consistency in the

[13]A property is semi-decidable if algorithms can be constructed that are guaranteed to report the respective property after finite (but indefinite) time if applied to a set that actually has this property, but possibly run forever otherwise.

[14]It is shown ([Aho, et al., 1974]), [Papadimitriou and Steiglitz, 1982]) that even the "satisfiability problem" of Boolean formulas — whether a Boolean formula can be made true by some truth assignment to its variables — is NP-Complete.

E-R model).

2.4 SIC Reformulation and Decomposition

The reformulation and decomposition of a general SIC into several sub-SICs is related to previous work on efficiently checking SICs, an important research topic in the database literature. A naive approach is to perform the modification and then check whether the new database state satisfies all SICs. Such an approach is called *full integrity constraint checking* ([Ling, 1986]) or *total integrity checking* ([Nakano, 1983]).

Full SIC checking is time-consuming. Researchers have proposed a number of more efficient SIC checking techniques or algorithms. For example, Stonebraker [1975] considers monitoring immediate static SICs and elementary database updates by applying the "query modification" technique. Niacolas [1982] and Kobayashi [1984], respectively, present a simplification algorithm that transforms static SICs into simplified forms. Hsu and Imielinski [1985] provide a simplification method for transactions, and also for SICs expressed in the prenex normal form of relational tuple calculus. Ling and Rajagopalan [1984] propose a method for eliminating avoidable checking of integrity constraints expressed in first order predicate calculus. Ceri and Widom [1990] present a labelling algorithm to derive automatically the set of operations that may cause constraint violation for any given SIC expressed in a SQL-based language.

Basically, the above efficient algorithms are derived from the syntactic structure of a SIC specification. Some researchers (e.g., [Qian and Wiederhold, 1986], [Qian and Smith, 1987]) go further to propose transformational mechanisms that exploit knowledge about the application domain and even database physical organization structure to reformulate SICs into semantically equivalent, but more efficient ones. Bernstein et al. [1980] has

proposed improving SIC checking efficiency by maintaining some redundant data (e.g., minima and maxima of certain sets).

All the above work is based upon the assumption that the current database state satisfies all SICs that have been specified. This type of checking is called *incremental integrity constraint checking* ([Nakano, 1983], [Ling, 1986]).

The purpose of previous work on SIC decomposition is mainly to make SIC checking more efficient. Some of them only attach a set of operations, which may cause constraint violation, to a SIC. Others rewrite the original SIC into a set of sub-SICs that is semantically equivalent to the original SIC. However, one should note that when a general SIC is decomposed into sub-SICs, it is possible that the sub-SICs might have different violation actions. Thus, these sub-SICs are also part of database specifications rather than just for implementation efficiency.

In addition, the previous algorithms are only suitable for few types of SICs in some restricted languages.

2.5 Automated Database Design Aids for Eliciting SICs

Yang and Goldstein [1989] survey twenty automated database design systems to investigate how SICs have been included. Many systems (e.g., I^2S [Kawaguchi et al., 1986]; SA-ER [Carsnell and Navathe, 1987]; ACME [Kerstern et al., 1987]) do not attempt to identify SICs at all. Some systems (e.g., SECSI [Bouzeghoub and Gardarin, 1984; Bouzeghoub et al., 1985; Bouzeghoub and Metais, 1986]; RIDL* [Troyer, 1989]; E^2R [Kozaczynski and Lilien, 1988]; CHRIS [Furtado et al., 1988]; Gambit [Brägger et al., 1984]; EXIS [Yasdi and Ziarko, 1987]; TSER [Hsu et al., 1988]; PROEX [Obretenov et

al., 1988]; Modeller [Tauzovich, 1989]; EDDS [Choobineh, 1985; Choobineh et al., 1988]; OICSI [Rolland and Proix, 1986; Cauvet et al., 1987; Proix and Rolland, 1988]; and Database Generation Tool [Maryanski et al., 1984; Maryanski and Hong, 1985]) do provide some mechanisms for eliciting and representing SICs. However, only a few types of SICs are identified. Most of them are the common SICs, e.g., incidence constraints, totality constraints, or inherent to data abstractions. No system provides a model to guide the incorporation of SICs.

The SIC representation of some systems is as arcs in a semantic network (e.g., OICSI) in logic (e.g., Gambit), or rules (e.g., CHRIS, EDDS). Some SICs are even represented as preconditions/post-conditions of transactions or events rather than data SICs (e.g., E^2R, EXIS). Some SIC features mentioned above are not explicitly represented. Gambit may be the one that represents them most explicitly. However, in Gambit, the responsibility for knowing the operation type on which a SIC must be checked rests with the database designer.

The discussions of SIC verification reported for these systems are either nonexistent or tend to be somewhat rudimentary.

Chapter 3

A Model for Representing Semantic Integrity Constraints

This chapter presents a language, called **the SIC Representation model**, to represent the features of a SIC that are mentioned in Section 2.1.9.

3.1 An Overview of the Representation Model

The SIC Representation model integrates and formalizes the ideas of Fernandez et al. [1981], Bertino and Apuzzo [1984], and Date [1983]. This model specifies precisely features of a SIC. The original model proposed by Fernandez et al., and Bertino and Apuzzo has been extended to include a measure of certainty, and some of their original concepts have been modified. The *cursor* concept in UDL proposed by Date has also been incorporated into the model.

The model represents a constraint in terms of six components: **Object** (**O**), **Operation Type** (**T**), **Precondition** (**C**), **Predicate** (**P**), **Certainty Factor** (**F**) and **Violation Action** (**A**). In addition, each constraint is given a descriptive name. The justification of including these six components is provided in Chapter 4. This chapter only gives the description of the model.

Using this model, a SIC is represented as:

SIC-Name

CERTAINTY	**F** (Certainty Factor)
FOR	**O** (Object)
ON	**T** (Operation Type)
IF	**C** (Precondition)
ASSERT	**P** (Predicate)
ELSE	**A** (Violation Action)

In terms of the usual production rule syntax, the above whole statement (except for *SIC-Name*) is interpreted as:

with certainty **F**
IF **(O,T,C)** *THEN*
(IF NOT **P** *THEN* **A)**

SIC-Name is used as an identifier that conveys some meaning of a SIC. It is not essential to the Representation model.

The following example, named as *project_employee_minimum_salary* for convenience, will be used for illustration. Suppose that there is an *Employee* entity, and a *Work_for* relationship in the E-R model and the corresponding *Employee*, and *Work_for* relations in the relational model[15]. A SIC might state "*if an employee works for any project, his (her) salary should be greater than $10,000*". This SIC can be decomposed into several sub-SICs represented in the Representation model. Each sub-SIC is operation-dependent and only relevant to a single object. In addition to the related sub-SICs on the update of *Work_for*'s primary key, two sub-SICs are:

[15]That is, a separate relation, *Work_for*, is assumed for simplifying the comparison of our model to other work in the literature since they only deal with the relational model.

— one for *Employee.Salary* on update;

— one for *Work_for* on insertion

The first one may be represented as the following.

Employee.Salary-U-RshipDepEntVal-(Work_for)

CERTAINTY	certain
FOR	Employee.Salary
ON	update
IF	∃ Work_for, rship_occ_part(Work_for, "Employee", Employee)
ASSERT	Employee.Salary > 10000
ELSE	reject

Interpretation: The first line is the SIC name. It indicates that this is a SIC for *Employee.Salary* on update and it is a *"RshipDepEntVal"* type because it asserts that the existence of a relationship (*Work_for*) depends on the value of an entity attribute (*Employee.Salary*). The *rship_occ_part* is an assertion that stands for "relationship occurrence participant". In this example, it is used to specify the *Work_for* occurrence in which the currently checked *Employee* occurrence participates with the entity type, "Employee". This SIC states that with 100% certainty, when an *Employee.Salary* occurrence is to be updated, if the *Employee* participates in at least one *Work_for*, his or her *Salary* must be greater than $10,000. Otherwise, the update operation is rejected. Note that the database designer might choose "propagate(delete(Work_for))" as the violation action. In that case, the propagation action might imply that if the organization could not afford the minimum salary for an employee, it

could not require that the employee be associated with any project (so the current *Work_for* occurrence must also be deleted).

The restriction intention of a SIC represented by this model is expressed as the predicate component (**P**) separately. Its other components precisely specify the SIC features listed in Section 2.1.9. These are:

1. certain or uncertain: indicated by the component **F** — certainty factor;
2. applied data: shown in the component **O** — object;
3. operation-dependent: a SIC represented in this model is always operation-dependent, the operation is specified in the component **T** — operation Type.
4. unconditional or conditional: the conditions are listed in the component **C** — precondition;
5. strong, soft or self-correcting: indicated by the component **A** — violation action.

A SIC represented in this model is always dynamic in that it indicates a valid database state transition in which the object would be manipulated by the specified operation type. A static constraint is represented by rewriting it into one or more dynamic constraints for the related object(s) on the operation(s) that could cause unallowed database state(s). The reasons for doing this are as follows:

- Because some constraints are inherently dynamic, this allows uniform representations for all kinds of constraints.
- Enforcement efficiency can be greatly increased by knowing which operations can cause constraint violations.

- Violation actions may be different depending on the operation types causing constraint violations.

The usual approaches to representing a SIC mix up several components in one statement and do not explicitly describe the above features. Taking a SIC represented in a traditional language, one could only assume that it is 100% certain; related to all objects mentioned in that SIC; applicable to all primitive database access operations; unconditional in all contexts; and so strong that it would cause errors if violated. For instance, according to the language used by Ling and Rajagopalan [1984], the above example would be expressed in the prenex normal form of the first order predicates as follows.

E ranges over Employee

J ranges over Work_for

$\forall$ E $\forall$ J (E.EmpNo $\neq$ J.EmpNo OR E.Salary $>$ 10000)

Evaluation:

This statement is neither precise nor powerful because of the following:

(1) It is not clear how certain this SIC would be. So, it can only be assumed as 100%.

(2) Since *E* and *J* appear there and no operation type is explicitly specified, it would seem that the SIC is relevant to any operation on an *Employee* or a *Work_for*. However, in fact, it could never be violated by deletion or insertion of an *Employee*, or deletion of a *Work_for*.

(3) It does not clearly distinguish between the condition specifying those employees to whom the constraint would apply (those working for some project(s), i.e., the condition "E.EmpNo = J.EmpNo") and the assertion

on the salary of those employees (i.e., the expression "E.Salary > 10000").

(4) Since no violation action is specified, it can only be assumed as "reject".

The representation in the SIC Representation model is also more precise than the original models presented by Fernandez et al. [1981], and Bertino and Apuzzo [1984]. According to the model presented by Bertino and Apuzzo [1984], the above example would be expressed by a single SIC in which the object component may include "*Employee*" and "Work_for" and the operation component may include "update" and "insert". It implicitly means that the SIC is asserted on the insertion of an *Employee* or *Work_for* and the updates of any attribute of an *Employee* or *Work_for*.

In addition, the representations in terms of the Representation model have some advantages that will be described in detail in the next chapter. In short, the certainty factor introduces fuzzy semantics, facilitates knowledge-based query processing and provides deductive capabilities. The identified object and operation type are useful for efficient enforcement and together with the violation action, they provide different perspectives for specifying and enforcing a general SIC. The separation of precondition from predicate allows natural and precise representation of a SIC. The explicit representations of the object, operation type and precondition components are useful for applying SIC abstractions (that are introduced in Section 4.2).

These six components and the naming convention of the SIC Representation model are described in the following sections. Detailed BNF (Backus-Naur Form) descriptions of the model are given in Appendix A. The rest of this section describes briefly the notation and the allowed expressions in this language.

Expressions First order logic and higher order logic expressions, arithmetic expressions and date expressions, are allowed in the precondition and predicate components. They may consist of quantifiers — $\forall$ (for all), $\exists$ (there exists), logical connectives — "$\wedge$" (conjunction), "$\vee$" (disjunction), "$\neg$" (negation), but not the "$\rightarrow$" (implication) connective since it has been separated as the precondition. They may also contain aggregate functions: e.g., *avg, count, max, min, sum*, or other user-defined aggregation functions. In addition, a special pair of functions, *old* and *new*, are used. The *new/old* function in SIC references the new/old value of the referenced object after/before checking the SIC. An implemented integrity maintenance subsystem of a DBMS must assure the correct functioning of the *new* and *old* functions[16]. One set expression is also allowed: *set*$\{E_1|$ *some restriction(s)*$\}$. This is read as *for all* E_1 satisfying "*some restriction(s)*"[17]. The special predicates used with the SIC Representation model are summarized in the Appendix B[18].

Subscript Notation This research uses a subscript to simulate the "cursor" in Date's UDL to emphasize that a SIC is enforced on an occurrence although it is specified intensionally for an entity/relationship/relation type. A *cursor* in UDL is an object whose value is (normally) the address of some specific record in the database ([Date, 1983]). Since a SIC is enforced on an occurrence, if there are other occurrences of the same entity,

[16]If the SIC is checked immediately, it references the new/old value of the referenced object after/before the operation. However, if the SIC is deferred to be checked at end of transaction, it references the new/old value of the referenced object after/before the transaction.

[17]The keyword *set* can be omitted. This notation is adopted from an "integrity constraint definition language" [Gardarin and Melkanoff, 1979].

[18]In addition, this research follows the Prolog naming convention that a variable is written as a word beginning with a capital letter and an atom is written as a word beginning with a lower case. The only exceptions are keywords (CERTAINTY, FOR, ON, IF, ASSERT and ELSE) and the SIC name in the first line, which are character strings in nature, not variables.

relationship or relation type referred in the precondition or predicate component, different subscripts will be used to distinguish them. The one to be checked is referenced by attaching a default subscript 0, e.g., E_0. Variables with subscripts other than 0 (e.g., E_1) represent any occurrence of the same object (e.g., E) type. In terms of programming language, the one with subscript 0 refers to the particular inserted/deleted/updated "value" to be checked; those with subscripts other than 0 are "variables" of the same object type. If there is only one occurrence (i.e., the one to be checked) referred in the precondition and predicate components, the default subscript 0 is omitted. In any case, we do not use a subscript in the object component because the variable in that component is used to indicate that the SIC is applicable to any occurrence of the specified type.

An example from Date [1983] will help to clarify the above subscript idea. Suppose that there is a relationship *Supply* in the context of "*Supplier Supply Part*"; and a SIC requiring that "*any quantity value of a supply must not be more than 5 percent greater than the average of all such values*". One of the required SICs is represented by the Representation model as below, in which $Supply_1.Qty$ stands for any *Supply.Qty* occurrence. The assertion in its predicate component requires that each of other *Supply.Qty*'s must satisfy the average property when an occurrence is to be deleted.

Supply-D-AggFcn-(Supply.Qty)

CERTAINTY	certain
FOR	Supply
ON	deletion
ASSERT	$\neg(\text{Supply}_1.\text{Qty} > 1.05 \times \text{avg}(\{\text{Supply}_1.\text{Qty} \mid \text{Supply}_1 \neq \text{Supply}_0\}))$
ELSE	reject

Legend: Supply_0 indicates the value of Supply occurrence currently checked.

3.2 SIC Name

SIC name is not counted as a component in the Representation model since it is not essential. A database designer may freely use other naming conventions, such as the simple one *I1, I2, ...* used in the literature ([Bertino and Apuzzo [1984], and [Date, 1983]), or some application dependent conventions. In this research a concatenated character string is used as a SIC name for conveying some of the meaning of a SIC and also for serving as its unique identifier. This convention allows the *SIC elicitation subsystem* of an automated database design system (introduced in Chapter 7) to generate a SIC name automatically by incorporating an abbreviated application domain-independent SIC type and some application information (object type, operation type and related object type set). A SIC name contains five parts. The general format is like:

ObjectType-OperationType-SICType-(RelatedObjectTypeSet)-SequenceNo

For example, from "*Employee.Salary-U-RshipDepEntVal-(Work_for)*" we know that this SIC should be checked on the update of *Employee.Salary* because the existence of a *Work_for* relationship occurrence depends on its value.

1. The *Object Type* (e.g., *Employee.Salary*) is the specific type name of an attribute, entity, relationship, or relation for which this SIC is asserted. It corresponds to the component **O** in the SIC.

2. The *Operation Type* is either *U, I*, or *D* representing update, insertion, or deletion, respectively. Similarly, it corresponds to the component **T** in the SIC.

3. The *SIC Type* is the abbreviation of a SIC type conveying some application domain-independent meaning; e.g., *"Totality"* (if an entity occurrence exists, it must participate in some minimum number of relationship occurrences of the specified type), *"RshipDepEntVal"* (the existence of a relationship depends on an attribute value of an entity type). By applying the E-R-SIC model (introduced in Chapter 5), we can classify SICs into a number of domain-independent SIC types.

4. The *Related Object Type Set* includes those object types that appear in the pre-condition and predicate components of a SIC and can supplement the meaning of the SIC type. A SIC name may not have this part if there is no such objects. For example, there is no other object type related to an absolute cardinality constraint for an entity type, which restricts the maximum number of occurrences of the entity type that can exist in a database. However, in the above *project_employee_minimum_salary* example, it may be useful to know which relationship type (i.e., *Work_for*) depends on the attribute (i.e., *Employee.Salary*). For domain constraints on the insertion of an entity, although all the attributes of the entity are referenced in the SIC, they need not be included in its SIC name since it is implied by the SIC type *"Domain"*.

5. The *Sequence No* is a positive integer number. In a complicated application, it is possible that the above parts would not be sufficient to distinguish one SIC from others. In that case, as a last resort, a sequence number starting with 1 will be added.

3.3 Certainty Factor (F)

A certainty factor, **F**, which does not appear in Fernandez et al. [1981] and Bertino and Apuzzo [1984], is introduced in the Representation model. A certainty factor is closely related to the predicate (**P**) and violation action (**A**). It may be attached to SICs in the following alternative ways:

1. **Ratio Scales:** A ratio scale to measure the certainty of a SIC is needed if we would use the certainty factor to provide deductive capabilities under uncertainty. The certainty factor is defined to have a value between 0% and 100%. Any SIC with a certainty factor less than 100% can only have "warning", "conditionally_reject", or "conditionally_propagate" as its violation action. The value of the certainty factor is based on the experience of the database designer. If the value is too low, it implies that the SIC is very unlikely to hold in the database and is less useful in providing semantic information.

2. **Ordinal Scales:** If we are only interested in traditional databases and there are a number of uncertain SICs, an ordinal scale to measure the certainty of a SIC is needed. This ordinal scale would be used to rank the certainty of SICs in the same "family" (in terms of the same restriction intention, but different levels of stringency) when verifying and enforcing them. The certainty factor might be defined to have a value between 1 and 10 if there are at most 9 uncertain SICs for the same "family" of SICs. Any SIC with a certainty factor less than 10 is an uncertain SIC.

3. **Two levels:** It is possible that a database designer would like to specify at most one uncertain SIC for a restriction intention because an organization is mainly

concerned with SICs with 100% certainty. In this case, two discrete levels of certainty are enough. In this case, "uncertain" (i.e., certainty $\leq$ 100%) would have a "warning", "conditionally_reject", or "conditionally_propagate" violation action whereas "certain" (i.e., certainty = 100%) would have a corresponding "reject", or "propagate" violation action.

4. **Fuzzy terms:** It is also possible to attach a fuzzy term, e.g., "usually", "sometimes", as the certainty factor if that term has been properly specified to be equivalent to a specific certainty ratio number (e.g., "usually" may be equivalent to 80%).

3.4 Object (O)

O represents the data object to which the SIC applies. In the E-R model, it corresponds to any occurrence of the specified attribute, entity, or relationship type. In the relational model, it corresponds to any occurrence of an attribute or a relation type, i.e., a column, or tuple. The $E.A$ or $R.A$ is used to refer to the attribute A of entity E or relationship (relation in the relational model) R. Note that an attribute is treated as an object in this model since it would make the SIC clearer and it is possible that a constraint is relevant to an update of an attribute, but not the insertion/deletion of its related entity, relationship, or relation. For example, in the above *project_employee_minimum_salary* example, it is not relevant to the insertion or deletion of an *Employee* although it is relevant to the update of an *Employee.Salary*.

3.5 Operation Type (T)

The operation type, **T**, (also called the access type [Bertino and Apuzzo, 1984]) specifies the types of database manipulation operations to which the SIC is applicable. Only three operation types need to be considered — **insertion, deletion, or update**. Note that a SIC is never relevant to a retrieval (query) operation.

The data definition operations, i.e., "creation" or "destruction" of an entity, relationship, or relation type, are also excluded[19]. A database designer should design a database to include those necessary object types according to the information requirements. However, whether an object type should be in a database schema is not a SIC. Destroying an object type is concerned with the database reorganization. In the case of database reorganization, destroying an object type may have different implications in different situations. For example, destroying a type may occur because this type ceases to exist in the real world or just because the organization will no longer keep the super-type (e.g. *Person*) information, but is still interested in its subtypes (e.g. *Employee*). The database designer needs to be involved to discover the changed information requirements. The different situations cannot be foreseen and automated in the database design phase.

3.6 Precondition (C)

The **precondition (C)** specifies the implicit or explicit presuppositions of a SIC. If it is not satisfied, it would cause the predicate of the SIC to be either undefined or irrelevant. For example, in the above *project_employee_minimum_salary* example, the SIC for *Employee.Salary* on update is irrelevant to any employee who does not work for

[19] So the language represented by this model is not a relationally complete query language.

any project. It also identifies the object (**O**) in the context of this SIC. In the above example, the SIC for *Work_for* on insertion would have the precondition as "∃ Employee, rship_occ_part(Work_for, "Employee", Employee)" to indicate the *Employee* occurrence participating in the specific *Work_for* occurrence. If a SIC has no precondition, the keyword *IF* may be omitted.

3.7 Predicate (P)

The **predicate (P)** is the assertion for the database state transition when the object (**O**) is to be manipulated by the operation (**T**). If it is true, the operation (**T**) on the object (**O**) is allowed to be performed. For an attribute other than primary key[20], the predicate should involve the attribute itself. For an entity, relationship, or relation, the predicate involves its attributes, aggregate attributes (e.g., average value), or related other entities and/or relationships (relations in the relational model). Since a SIC places logical restriction on data, its predicate component should contain invariant assertions on the related objects rather than procedural statements of programs.

The precondition and predicate are both expressions that control whether the violation action will be invoked. The precondition specifies the general state of the database that makes the constraint relevant. The predicate specifies something that must be true about the object after the operation. The justification of having both components is described in detail in Chapter 4.

[20]We have this exception because in the relational model and traditional E-R model, primary keys play the role of "surrogates". The update of a primary key of an entity/relationship/relation may imply the deletion of an old entity/relationship/relation followed by the insertion of a new one.

3.8 Violation Action (A)

The violation action (**A**) specifies how the system is to behave if the predicate P is false when the SIC is checked. In order to have precise meaning for a single SIC, it is desirable to have a single violation action (or a deterministic series of well-defined actions) rather than a complex procedure with many choices. The violation action may be specified as follows.

1. *"warning"* — allow the access operation, but issue a warning.
2. *"reject"* — reject the access operation, signalling an error.
3. *"propagate"* — allow the access operation, but perform a related corrective action to restore the database to a correct state.
4. *"conditionally_reject"* or *"conditionally_propagate"* — request confirmation from the user to ignore the SIC; the processing of the access operation is suspended until the user indicates either to ignore the SIC or to take the violation action (reject the access operation, or allow the operation but propagate to perform a corrective action).

Discussion on Propagation Traditionally, the violation action of a SIC is just a rejection. A warning action indeed turns off the enforcement of the SIC except for generating a message. A propagation action changes other objects in the database. Date [1983] indicates that integrity rules may be considered important special cases of *triggered procedures*[21]. In discussing semantic integrity constraints, Fernandez et al. [1981] mention the following problems raised by the trigger mechanism.

[21] Other application areas for triggered procedures include *virtual fields, security*, etc. [Date, 1983]

(1) Integrity or security violation within auxiliary procedures could result in a never-ending sequence of invoking procedures.

(2) The proper access rights of the invoked procedures are an issue: Should these be the rights of the invoker of the triggering maintenance operation or the rights of the DBA who specified the auxiliary procedure?

(3) Triggering of maintenance procedures results in a somewhat confusing division of responsibility between the user (or the programmer) and the DBA. For example, who should update the count of employees on inserting a new employee? The system or the programmer?

The problem of security or access rights is beyond the scope of this research[22]. Besides, as stated before, the intention of this research is not to propose an approach to replacing the transaction or application programming. If the intention of the violation action is just to maintain a correct database state, there is no reason that we would need a complex violation action. Therefore, although in principle a "violation action" could be very complex, in this research it is generally a single action or a deterministic series of well-defined actions. The responsibility of programmers to write proper application programs to update data is not changed[23].

If the violation action is to propagate to insert some other objects, establishing the attribute values of those objects might become a problem. It is not possible for a database designer to know all these values during database design. Those values that cannot be known by an automated database design system will be assumed to be "null" (i.e., "unknown"). This may violate other SICs if some unknown attribute values (e.g., part of

[22]However, this unsolved security problem raised by Fernandez et al. [1981] might also be one argument against complex violation actions.

[23]However, now a DBMS becomes more powerful to detect programming errors of programmers and takes violation actions. Programmers need not check SICs because the DBMS will do that.

key of a relationship) are not allowed to be "null". There are two approaches to dealing with this problem. In the first approach, the violation action "reject" rather than *"propagate(insert(O1))"* must be specified. In the second approach, the propagating insertion is allowed. Compared to the first approach, the second approach is more powerful. Even if the propagating insertion is finally "rejected" due to other SICs, it is easier for the user to understand why the original insertion operation is rejected. It also leaves open the possibility of designing an enhanced integrity maintenance subsystem of DBMS that will prompt the user to supply the unknown attribute values or obtain them from some other sources ([Casanova and Tucherman, 1988]).

Two-valued Logic Note that the violation action is taken only when the predicate, not the precondition, is false. What would happen if a database allows null values? A null value can have at least two interpretations: *"unknown"* and *"not applicable"*. In this research, "*null*" is only allowed to represent "*unknown*" because the "*not applicable null*" should not appear in a well-defined, properly-normalized database with clear semantics[24]. In principle, some warning messages can be issued[25] if the predicate is evaluated to be unknown. However, since the "unknown" values will eventually become "known", they will be checked at that time. If they violate SICs, appropriate actions will be taken

[24]The existence of *"not applicable null"* in some attributes of an entity, relationship, or relation indicates that these attributes do not apply to all occurrences. It implies that some subtypes should be specified in order to clarify the semantics. In the literature (e.g., [Lee, 1988]) there are two other interpretations of "null". Lee states that, for example, with regard to John's spouse, the null value may mean: (1) literal interpretation *"null"* (e.g., the spouse name is "null"); (2) sense of "none" (e.g., John is a person, but he has no spouse); (3) "not applicable" because spouse is not an attribute of the object (e.g., John is the name of a building, not a person); (4) "unknown". Note that the first interpretation is the result of bad implementation of "null". "Null" should be represented by an unambiguous symbol in a database. The second and third interpretations are also results of a bad database design — the semantics are not clear because some attributes are not defined for some occurrences of the specific "type". In the above example, there should be two types, *Person* and *Building*, to avoid the third interpretation. The subtype *Married_Person* should be defined to avoid the second interpretation. In this research, the *"not applicable null"* interpretation includes both the above second and third interpretations.

[25]These are termed "weak violations" by Ho [1982].

then. Therefore, in this research, for simplification, it is assumed that if the predicate is evaluated to be unknown, no violation action will be taken immediately. By taking such a position, the verification of SIC consistency can be simplified from 3-valued (true, false, unknown) to 2-valued (true, false) logic.

Chapter 4

The Application of the SIC Representation Model

This chapter describes how we can apply the SIC Representation model for database design and management. In Section 4.1 it is claimed that by applying this model we can represent precisely the features of any SIC. In Section 4.2 SIC abstractions are introduced to facilitate SIC specification and management. The usefulness of the Representation model in SIC abstraction is also discussed. In Section 4.3 the application to database management is briefly described.

4.1 Completeness of a SIC Specification for Database Design

Su & Raschid [1985] and Shepherd & Kerschberg [1986] suggest that "constraints" must explicitly specify both declarative semantics as well as process oriented or operational semantics. *Declarative semantics* correspond to logical formulas describing relationships between objects in a database. The information needed to check that these relationships are true or to maintain these relationships is the *operational semantics.* It is also suggested that the "constraint" formalisms must provide information along the lines of WHAT, WHEN, WHERE and HOW if they are to be complete. Although our SICs are only a proper subset of their "constraints" (refer to Section 1.3), the same formalism with some modifications can be applied to examine the completeness of a SIC specification.

1. **Operational and Declarative Semantics.** In the Representation model, the predicate (**P**) and certainty factor (**F**) components specify the invariant declarative semantics among data objects. The operation type (**T**) is relevant to the operational semantics to check these invariant facts. The violation action (**A**) specifies the operational semantics that are needed to maintain these assertions. In addition, the object (**O**) and precondition (**C**) components supplement both declarative and operational semantics to assert and check these invariant assertions. The certainty factor (**F**) also provides some operational semantics since the highest certain SIC should be enforced first.

2. **WHAT, WHEN, WHERE and HOW.** The Representation model specifies:

   - WHAT the SIC requires — its invariant assertions (predicates, **P**) and certainty (certainty factor, **F**).
   - WHEN the SIC is to be checked — these assertions should be checked when some primitive access operation (**T**) (update, deletion, insertion) is performed on the data.
   - WHERE the SIC occurs — these assertions are applicable to some data object (**O**) in some contexts (preconditions, **C**).
   - HOW the system should behave in the case of these assertions being violated (violation action, **A**).

Therefore, the Representation model provides declarative and operational semantics of a SIC specification. It also provides a SIC specification along the lines of WHAT, WHEN, WHERE, and HOW. The rest of this section further explores whether we would exactly need these six components in order to represent completely declarative and operational semantics of a SIC specification and express precisely its features — can we have

fewer or do we need more?

4.1.1 Not Fewer Components

The strengths of the Representation model are that it is precise enough to express the features of a SIC; uniform for all kinds of SICs; and powerful enough to offer a corrective action rather than simply reject the operation violating the SIC. As mentioned in the previous chapter, traditional languages do not explicitly describe all features included in the Representation model. A SIC represented in a traditional language could only be assumed as 100% certain, related to all objects mentioned in that SIC; applicable to all primitive database access operations; unconditional in all contexts; and so strong that its violation would cause errors. It is beyond question that the predicate (**P**) component is absolutely needed. Other components need to be further discussed.

Object (O) Depending on the language used to represent a SIC, a number of objects may be mentioned. However, the declarative and operational semantics of the constraint may not be relevant to some of these objects. In the *project_employee_minimum_salary* example of Chapter 3, the constraint is not relevant to the *Project* that may be mentioned if it is declared in a natural language. Neither is it relevant to the *Employee* if it is represented in a pure first order logic such as the one used by Ling and Rajagopalan [1984]. Consider another example. Suppose we have a constraint: "*if an employee works for any project, his (her) salary cannot decrease*". This constraint is only relevant to the *Employee.Salary*, not *Employee*, *Project* or *Work_for*. In order to have precise declarative and operational semantics for a SIC, it is necessary to specify explicitly for which object this SIC is asserted. Since in this model a SIC is applied to a single operation, which can only manipulate a single object, it is sufficient to assert only for this object.

Operation Type (T) If a constraint is relevant to an entity, relationship, or relation, but not an attribute, and is operation-dependent, the explicit specification of its operation type is needed. If the constraint is operation-independent, there are still two advantages (in addition to improving enforcement efficiency) to specifying the operation type:

- First, explicitly expressing which operation could cause constraint violation provides the valuable operational semantics and clarifies the restriction intention (declarative semantics) of the SIC. It would be desirable to have a conceptual picture of the consequence of constraint enforcement even in the database design phase. It would also be helpful for later designing transaction specifications. For example, given the constraint "*the total number of employees cannot exceed 200*", it is helpful to know that its restriction is only relevant to the insertion of an *employee* although the original constraint is operation-independent.

- Second, it is impossible to specify the violation action without referring to a specific operation type. When a general SIC is rewritten into several sub-SICs for separate objects in terms of the Representation model, their objects, operations, and possibly different violation actions provide useful declarative and operational semantics — necessary conditions for operations on the objects in the object component and sufficient conditions for propagated operations on related objects.

However, for a SIC asserted on an attribute, the explicit operation type specification is redundant since update is the only possible manipulation operation type. In this case, we explicitly specify it only for clear and uniform representation.

Precondition (C) One may argue that it is unnecessary to separate precondition from predicate in terms of pure logical expressiveness. However, without this separation, the declarative and operational semantics of the SIC are ambiguous. The precondition specifies the current database state that makes the constraint relevant; the predicate asserts the allowable state after the operation on the object. We have seen the *project_employee_minimum_salary* example in Chapter 3. Some constraints are more naturally expressed in the *IF ... THEN ...* format. By explicitly separating the predicate from the precondition, the restriction intention of the predicate becomes clearer and is closely related to the violation action that now can be simplified to be a deterministic series of well-defined actions (usually a single one).

Even if a constraint is not in the *IF ... THEN ...* format when we first write it in English, it may have some implicit "presuppositions". For example, in a database containing *Department*, *Employee*, and *Manager*, a constraint might be stated in English as *SIC-0*[26]: *"each employee must earn less than his (her) department manager"*. In that database, the above statement has two implicit presuppositions: (1) *"the employee belongs to some department"*; and (2) *"the department has a manager"*. If these presuppositions are only true for some employees and departments, *SIC-0* is conditional and the presuppositions should be explicitly represented in its precondition component. If these are facts that are true for all employees and departments, we should have two other SICs, each of them asserting one of the above facts in its predicate component. In addition, these presuppositions should also be in the precondition component of *SIC-0* so that the connection between a specific *Employee* and *Manager* are clear[27]. Each SIC would have precise meaning.

[26] In this chapter, *SIC-i*, *i=0,1,2,...* are used as SIC names for simplicity.

[27] That is, by these preconditions, we can find the manager occurrence related to an employee or find the employee occurrences related to a manager. However, these are not the restriction intention of the above SIC.

The precise context specifications in the precondition are also needed to provide operational semantics when enforcing some related SICs. Suppose we have two SICs for *Employee.Salary* on update.

SIC-1

CERTAINTY certain

ASSERT Employee.Salary $\leq$ 100,000

ELSE propagate(update(Department.Budget))

SIC-2

CERTAINTY certain

ASSERT Employee.Salary $\leq$ 150,000

ELSE reject

When updating the salary of an employee to be \$180,000, we would have an enforcement problem — which violation action should be taken. The problem is caused by the imprecise representation of *SIC-1*. The original restriction intention of these two SICs is: *"the salary of an employee should be less than or equal to \$100,000, otherwise, if it is not too high, we can increase the department budget; but if it is higher than \$150,000, we can only reject it"*. The precondition of *SIC-1* should include an expression, "*Employee.Salary* $\leq$ *150,000*"[28].

[28] Alternatively, we can use an expression, "*checkpreSIC(SIC-2, Employee.Salary)*", that checks the pre-SIC, *SIC-2*, for *Employee.Salary* and returns true if the pre-SIC is satisfied. One SIC's (say *SIC-1*) precondition may be the precondition and predicate of other SICs (say, *SIC-2, SIC-3*). *SIC-2* and *SIC-3* may be called as the *pre-SICs* of *SIC-1*. If a data object violates the pre-SICs, *SIC-1* becomes irrelevant to further testing.

Violation Action (A) Without the violation action component, all SICs could only be assumed to be strong. The SIC specifications would be less powerful since they could not include self-correcting or soft SICs.

Certainty Factor (F) If a database designer only considers certain SICs, he or she would not need the certainty factor component. However, the purpose of introducing certainty factors in database specifications is to represent more data semantics — including fuzzy semantics. If the certainty factor component was not included in the Representation model, all SICs would be assumed to be 100% certain — no exceptions. The violation action of a SIC would be either a rejection or a corrective action. Systems with only such SICs contain less declarative semantics since the fuzzy semantics are lost. They have also been criticized as too rigid to "*deal with unusual, atypical, or unexpected occurrences*", and "it is not possible to allow violations of integrity constraints to persist without turning off completely the checking of those constraints" ([Bordiga, 1985]). Having considered such possibilities and in order to avoid such criticisms, a database designer might specify too few SICs or too "generous" ones. The inclusion of certainty factors in the Representation model provides a mechanism for accepting data violating the assertions of some uncertain constraints — the introduction of "controlled inconsistency". For example, a database designer may specify a SIC for employee's salary as *"Employee.Salary* $\leq$ *$1,000,000"* even though few employees in an organization have salary greater than $100,000. Such a SIC would not catch common data entry errors, e.g., misplaced or omitted decimal points. Employing a certainty factor, he (she) might specify two or more SICs for the same object on the same operation. For example:

SIC-1: with 100% certainty, *Employee.Salary* $\leq$ *$1,000,000*

SIC-2: with 90% certainty, *Employee.Salary* $\leq$ *$100,000*

SIC-3: with 80% certainty, *Employee.Salary* $\leq$ *\$50,000*

SIC-2 and *SIC-3* are fuzzy (uncertain) semantic integrity constraints. If they are violated, different warning messages might be issued.

One may wonder since the certainty factor component relates closely to the violation action, we might only need the latter. However, we may lose some semantics if we only have the violation action component:

- Suppose that we only have at most one uncertain SIC for each restriction intention. It might be acceptable to keep only the violation action component if we can assure that the violation actions would correctly be specified and be consistent with their "implicit" certainty factors.

- Suppose that we have a number of uncertain SICs. In the above example, *SIC-2* and *SIC-3* should have measures of certainty at least on an ordinal scale. Otherwise, we cannot know whether these SICs are inconsistent[29]. In addition, when enforcing a set of uncertain SICs, some ordering is needed. The most certain SIC in the same "family" (in terms of the same restriction intention, but different levels of stringency) should be checked first since its violation is more likely to cause the database to be inconsistent, the second certain one is then checked, etc. If an occurrence violates the higher certain SIC, the remaining SICs with lower certainty in the same "family" need not be checked. Without the certainty factor, we would lose some declarative and operational semantics.

[29]If the *SIC-3* were with certainty 95%, *SIC-2* and *SIC-3* would become inconsistent. Note that SICs represented in this model are consistent if there exists a database state transition that is allowable with regard to all of the restrictions, i.e., considering all their components except for violation actions.

Conclusion Based on the above arguments, we can conclude the following:

> *If we wish to have uniform and precise representations for all kinds of SICs, it is necessary to have all six components of the Representation model.*

4.1.2 No More Components

Is it possible that more components are needed to represent declarative and operational semantics of a SIC or express its features? In the following, some possible proposals are discussed.

Enforcement Schedule Shepherd and Kerschberg [1986] suggest that a "constraint" formalism should include when a constraint is to be checked — after every transaction or at audit time. Note that their discussion corresponds to the classification based on the enforcement schedule described in the Section 2.1.7. As discussed in that section, the enforcement schedule is a transaction-driven specification. It is impossible to specify the enforcement schedule of a SIC as another component in the level of database schema.

Permanence As also discussed in the Section 2.1.5, the permanence of a SIC is not easily decided in advance since the organizational environment is changing. The permanence of a SIC is closely related to SIC maintenance rather than database design or SIC enforcement. If a SIC is still in a database schema, it should be enforced anyway. Its permanence information adds no more declarative or operational semantics. Therefore, this "permanence" information is not included in the Representation model.

Object Type, Set, or Occurrence The object component in the Representation model means that the SIC is applicable to any occurrence of the specified object type. Occurrences are fundamental things in the database, and insertion, deletion, and update are primitive operations on them. The integrity problems caused by data definition operations (destruction or creation) on data object types are not the focus of this research. Any SIC that involves data manipulation operations on higher levels of objects, e.g., types, or sets, is finally enforced at the occurrence level. Therefore, the explicit indication of the level to which the SIC is applied would add no more declarative or operational semantics.

Static or Dynamic As mentioned in Section 2.1.8, researchers have also discussed static and dynamic constraints. Should we explicitly specify a SIC as static or dynamic? As described in Chapter 3, the Representation model is indeed transition-oriented; that is, all SICs represented in terms of the Representation model are basically dynamic. The fundamental premise is as below.

Premise 4.1 *The current database state is semantically correct.*

A database state is semantically correct if it can be constructed starting from an empty database by a sequence of valid database primitive operations (insertions, deletions, or updates). A database operation is valid if the state transition caused by it satisfies the SICs that exist at the time the operation is performed. We assume that a database is initially empty. Over time, object occurrences are inserted into the database, then updated, and finally probably deleted. Before an occurrence is manipulated, the old database state is semantically correct. A SIC is specified so that the database transition caused by its primitive operation would bring the database to a new semantically correct state. Any

"static constraint" is represented by rewriting it into one or more dynamic constraints for the related object(s) on the operation(s) that could cause unallowed database state(s). By doing so, we do not lose its declarative semantics, rather, proper operational semantics are attached. It is unnecessary to indicate whether the original constraint is dynamic or static in this model.

In addition, a constraint on a sequence of operations is not a "real" SIC in terms of the SIC Representation model. Instead, it is the consequence of enforcing several SICs. For example, in order to model "*a car must be owned by a manufacturer before it can be owned by a dealer*", given that there are *Manufacturer_Ownership and Dealer_Ownership* relationships, and *Car, Manufacturer, and Dealer* entities, we would need four SICs:

Manufacturer_Ownership-I-RshipExclusive-(Dealer_Ownership)

CERTAINTY	certain
FOR	Manufacturer_Ownership
ON	insertion
IF	∃ Car, rship_occ_part(Manufacturer_Ownership, Car)
ASSERT	¬∃ Dealer_Ownership,
	rship_occ_part(Dealer_Ownership, Car)
ELSE	reject

Dealer_Ownership-I-RshipExclusive-(Manufacturer_Ownership)

CERTAINTY	certain
FOR	Dealer_Ownership
ON	insertion
IF	∃ Car, rship_occ_part(Dealer_Ownership, Car)
ASSERT	¬∃ Manufacturer_Ownership,
	rship_occ_part(Manufacturer_Ownership, Car)
ELSE	reject

Dealer_Ownership-I-RshipBeforeRship-(Manufacturer_Ownership)

CERTAINTY	certain
FOR	Dealer_Ownership
ON	insertion
IF	∃ Car, rship_occ_part(Dealer_Ownership, Car)
ASSERT	∃ Manufacturer_Ownership,
	rship_occ_part(old(Manufacturer_Ownership), Car)
ELSE	reject

Manufacturer_Ownership-D-RshipTriggerRship-(Dealer_Ownership)

CERTAINTY	certain
FOR	Manufacturer_Ownership
ON	deletion
IF	∃ Car, rship_occ_part(Manufacturer_Ownership, Car)
ASSERT	∃ Dealer_Ownership,
	rship_occ_part(Dealer_Ownership, Car)
ELSE	propagate(insert(Dealer_Ownership))

Interpretation:

These SICs restrict the relationships for a single *Car*. The first two SICs specify that *Dealer_Ownership* and *Manufacturer_Ownership* are exclusive. The third SIC restricts *Manufacturer_Ownership* to have existed at the time *Dealer_Ownership* is being created. Thus, these first three SICs require that if a *Dealer_Ownership* exists, then *Manufacturer_Ownership* must have existed in the past (and must no longer exist now). The fourth SIC requires that when a *Manufacturer_Ownership* is to be deleted, a *Dealer_Ownership* must be created. These four SICs together restrict *Manufacturer_Ownership* and *Dealer_Ownership* to exist in sequence. These SICs are independent of any transactions. However, because of them, the order of the related transactions is restricted. The sequence restriction on transactions is naturally guaranteed if we specify completely SICs on data. It need not be worried about at the level of transaction specification.

For example, let us consider the following four possible transactions. *Transaction-2* (which can be called *Transfer_Car* transaction to convey the application meaning) can only be performed after *Transaction-1* (which can be called *Produce_Car* transaction)[30]. Note that Transaction-3 and Transaction-4 are not allowed to be performed. *Transaction-3* would be rejected because either (i) if *Transaction-1* was not executed before, executing *Transaction-3* would violate the third SIC; or (ii) if *Transaction-1* was executed before, executing *Transaction-3* would violate the second SIC. *Transaction-4* would also be rejected due to the fourth SIC.

[30]However, the *Transaction-2* may never be performed — that is, a car may be always in the hand of a manufacturer. Whether the *Transaction-2* should be performed would depend upon the happenings in the real world. Also, note that since all these related SICs are enforced at the end of *Transaction-2*, we can switch the order of those two database operations inside it.

Transaction-1

BeginTransaction

insert(Manufacturer_Ownership)

EndTransaction

Transaction-2

BeginTransaction

delete(Manufacturer_Ownership),

insert(Dealer_Ownership)

EndTransaction

Transaction-3

BeginTransaction

insert(Dealer_Ownership)

EndTransaction

Transaction-4

BeginTransaction

delete(Manufacturer_Ownership)

EndTransaction

A SIC with explicit time restriction is still a data-driven constraint though it may involve a special system variable *Current_time* that registers the current clock time. For example, to model the SIC: *"an employee cannot receive a salary raise during his or her first 6 months in the company"*, an *Employee* entity must have a *HireDate* attribute, and the SIC will be:

Employee.Salary-U-TimeRestrictTransition-(Current_time, Employee.HireDate)

CERTAINTY	certain
FOR	Employee.Salary
ON	update
IF	Current_time < Employee.HireDate + "6 months"
ASSERT	new(Employee.Salary) ≤ old(Employee.Salary)
ELSE	reject

This kind of SIC may only occur in an environment in which the event that causes manipulation of the involved objects is processed in real time so that the *Current_time* in the computer matches the event time in the real world. Otherwise, *Current_time* must be replaced by a time-valued attribute that records the external event time, and the SIC becomes an ordinary data-driven constraint. In the above example, if the request to update *Salary* is not processed in real time, the expression in the precondition component would be: *"SalaryUpdateRequest.Date < Employee.HireDate + "6 months"*.

Conclusion Four possible proposals to incorporate more components in the Representation model have been discussed. They suggest either some things that cannot be specified in a database schema, or others that add no more declarative or operational semantics of a SIC. Given these six components we can represent any kind of SIC that is mentioned in the literature, and list its features precisely. These six components sufficiently provide the declarative and operational semantics of a SIC — what should be true in the database, and the information to check and maintain those assertions. Thus, we conclude the following:

It is sufficient to have these six components of the SIC Representation model to represent a SIC precisely.

With this SIC Representation model, the database designer can represent the data integrity semantics properly during conceptual modelling.

4.2 SIC Abstractions

One may be concerned that there would be a huge number of SICs represented in terms of the Representation model in an actual database. However, the explicit component separation in the Representation model allows us to apply abstraction concepts to reduce the number of SICs that must be specified using the full Representation model.

SIC Aggregation Assume that *O1* and *T1* are the data object and operation type for *SIC-1*, respectively; and *Oi* and *Ti*, where *i= 2,3, ...*, are the data object and operation type for *SIC-i*, respectively. *SIC-1* (called **aggregate-SIC**) is the aggregation of other SICs (called **component-SICs**) if

- *O1* contains all *Oi*'s as components;
- the operation *T1* on *O1* can be conceptually thought of as the combination of operations *Ti* on *Oi*;
- component-SICs and aggregate-SIC are sub-SICs decomposed from the same general SIC.

An aggregate-SIC may have its own assertions and violation action. The enforcement of an aggregate-SIC can be simulated by checking all of its component-SICs and its

own assertions. If an object violates any component-SIC, the violation action of the aggregate-SIC will be taken. Thus, we can use a special logical predicate (*checkcomSIC*, see Appendix B.2) to refer to all its component-SICs by their names in the aggregate-SIC and avoid having to write the same assertions explicitly for *O1* on *T1*. One example is that the domain constraint on insertion of an entity can be simulated by checking the domain constraint of updating its attributes (from unknown values to some values). We would have SICs asserting not-null, value, unique, etc., for each of its attributes on update. These assertions need not be repeated for the entity on insertion.

SIC Specialization Assume that *O1* and *T1* are, respectively, the object and operation type for *SIC-1*; and *O2* and *T2* are, respectively, the object and operation type for *SIC-2*. *SIC-2* is the specialization of *SIC-1* if *O2* is a specialization (i.e., sub-type) of *O1*, and *T1=T2*.

The specialized SIC would inherit assertions from its parent SIC with the proper variable substitution (*O2* for *O1*). Thus, representation of the specialized SIC can be omitted unless it has special restrictions. The specialized SIC's own assertions can only refine its parent SIC's assertions, but not overwrite them. For example, suppose that the database designer defines *SIC-1* for *Employee.Salary* on update: *"(Employee.Salary > 1000) ∧ (Employee.Salary < 120000)"*. The SIC representation for *Manager.Salary* on update is not needed unless there are special restrictions (e.g., *"(Manager.Salary > 15000)"*).

SIC Association A SIC is a set of other SICs if conceptually the enforcement of the **set-SIC** is the same as the enforcement of all of its **member-SICs** and nothing more. The violation action of the "set-SIC" is dummy because if an object violates any

member-SIC, the violation action of that member-SIC will be taken. Thus, we can use a special logical predicate (*checkmemSIC*, see Appendix B.2) to refer to all its member-SICs by their names in the "set-SIC" and avoid having to write the same representations of the member-SICs explicitly for the object asserted by the "set-SIC". The concept of SIC association may be useful for SIC enforcement in a DBMS. For example, all SICs for the same object on the same operation may be grouped as a "set-SIC" or several "set-SICs" according to their certainty factors and/or scheduling requirements. This kind of "set-SIC" is not a new type of SIC. However, in SIC specifications, conceptually, a SIC on updating the primary key of an entity (or relationship, or relation) can be simulated as two "set-SICs" that refer to SICs for deleting the corresponding old entity (relationship, or relation), and inserting a corresponding new entity (relationship, or relation), respectively.

Generic SICs By applying the above abstraction concepts, we can reduce the number of SICs that must be specified explicitly using the Representation model. The concept of generic SICs is introduced to reduce the number of required explicit representations even further. Assume we have the following *generic object types*:

— (1) *Entity** is the generalization (i.e., union) of all entity types that are defined by the database designer. *Entity*.Attribute** is the generalization of all attributes of all entity types that are defined by the database designer.

— (2) *Relationship** is the generalization of all relationship types that are defined by the database designer. *Relationship*.Attribute** is the generalization of all attributes of all relationship types that are defined by the database designer.

These generic object types are conceptual modelling objects. They do not actually exist in a database. That is, neither data definition operations nor data manipulation operations

will actually be applied to them. However, we can write some common SIC types (e.g., domain constraints) for them. These SIC representations for generic object types can be called *generic SICs* (SICs for specific object types can be called *specific SICs* in contrast). The precondition component of a generic SIC includes logical predicates[31] (refer to Appendix B.1) to indicate clearly the contexts where the SIC type is applicable (e.g., the fact that two entity types are exclusive) and/or identify the related information (e.g., its primary key, etc.) of the object type for which the SIC type applies. Suppose that we keep the constraint information for specific object types as logical predicates in the database (e.g., in the data dictionary). Also suppose that the DBMS would support SIC inheritance properly. Since all object types defined by the database designer are subtypes of these generic object types, if they satisfy the preconditions of some generic SICs, these SICs would be applied to them by the principle of SIC specialization. Thus, these generic SICs serve as "templates" for common SIC representations and are expected to be inherited by specific object types[32]. We would need only one representation for each SIC type (e.g., two entity types are exclusive) in a database regardless of the number of specific object types to which the SIC type applies. The advantage of this approach is to reduce the possibly huge number of explicit representations of SICs so that the conceptual structure and future maintenance (management) of SICs become easier. In addition, since generic SICs can be pre-defined in an automated database design system,

[31] These logical predicates in the precondition component of a generic SIC contain some uninstantiated variables. (For example, *Primary_Key* is a variable in the logical predicate *entity("Entity*", Primary_Key, ...)* These variables will be instantiated when a specific entity type (e.g., *Employee*) has the constraint information to satisfy the precondition and inherit the SIC. (For example, the above *Primary_Key* can be instantiated to be "EmpId" if the database designer has specified *Employee* as an entity type with the primary key "EmpId", that is, the assertion *entity("Employee", "EmpId", ...)* has been given.)

[32] The idea here is similar to the following simple case. In a database, there are entity types *Employee*, *Customer*, *Supplier*, etc. Though the *Person* entity type does not actually in the database, we can write some SICs for *Person*, which would be inherited by the specific entity types (e.g., *Customer*). The level of our *Entity** type is still higher than *Person*.

the consultation to elicit SICs would be more efficient. Section 7.3.1 will describe in detail how to apply generic SICs for representing some SIC types that are identified during conceptual modelling.

Usefulness of the Representation Model It is necessary to identify the object and operation type when applying SIC aggregation or specialization abstractions. The explicit component separation in the SIC Representation model helps identify these components easily. In addition, the representation of generic SICs relies heavily on the precondition component. Most SIC types (e.g., two entity types are exclusive) only apply to some specific object types. The precondition component of such a generic SIC indicates the conditions for the specific object type where the SIC type is applicable. A few SIC types (e.g., domain constraints) are common to all entity types. In that case, the precondition component of a generic SIC is used to identify the relevant object in this context so that its variables can be instantiated with proper values when a specific object type inherits the SIC.

4.3 Database Management

In this section, some possible applications of the SIC Representation model to database management rather than database design will be briefly mentioned.

4.3.1 SIC Management

A DBMS should include an integrity maintenance subsystem to enforce SICs. Some previous research (e.g., [Hammer and McLeod, 1975] and [Bertino and Apuzzo, 1984])

have proposed the functional structure of this integrity maintenance subsystem. Instead of fully describing its structure, this dissertation briefly discusses its major functions — enforcing and maintaining (adding or removing) SICs. With regard to SIC enforcement, the focus is on the basic checking strategy (the enforcement schedule) and on the violation actions.

SIC Enforcement If all accesses to the database are through transactions, the DBMS would enforce SICs through pre- and post-conditions on transactions. The integrity maintenance subsystem would only maintain SICs, and would not be responsible for enforcing them. If it is possible to access the database using primitive operations or if no pre- or post-conditions have been specified for a transaction, the integrity maintenance subsystem would determine which SICs must be enforced and when.

If the checking of a SIC is not at the end of the transaction, it should be done before the intended database operation is to be performed, not after it. While examining the assertions in the precondition and predicate components of a SIC, the effect of its specified operation type should be taken into account. The Bertino and Apuzzo's [1984] original criteria of deciding when a SIC must be enforced with regarding to a transaction are modified as below:

> Basically, a SIC is enforced at the end of the transaction, if the set of objects in all components of the SIC is affected (updated, inserted, or deleted) by more than one update, insertion, or deletion statement in the transaction. Otherwise, the SIC is enforced on each occurrence update, insertion, or deletion, if it is a SIC involving only one entity/relationship occurrence or tuple; or on each update, insertion, or deletion-request, if it is a SIC involving one or

more than one set of occurrences or tuples belonging to the same or different entity, relationship types, or relations.

The violation action of a SIC might become a part of a transaction. In determining the enforcement schedule of SICs, the related propagation actions should be considered too. For example, suppose a transaction includes *action-1, action-2, action-3, ...*, and suppose that *SIC-1* is only affected by *action-1* and has a violation action *violation-action-1*. Therefore, *SIC-1* is enforced immediately. If *action-1* violates *SIC-1*, *violation-action-1* becomes part of the transaction. If *SIC-2* is affected by *action-3* and *violation-action-1*, it should be enforced at the end of transaction. If the transaction is rolled back for any reason, the related propagation actions should also be undone and if a related propagation action aborts, the transaction must be rolled back too.

In most modern DBMS, there is a recovery mechanism that is responsible for recovering the database from many possible failures, e.g., transaction failures, system-wide failures, media failures ([Date, 1983]). We assume that the integrity maintenance subsystem cooperates with this recovery mechanism to keep information in a log required for undoing an operation request. The information needed in the log is similar to that for recovery, e.g., (i) identification of each modification (update, insertion, or deletion) statement; (ii) identifers of the occurrences affected by each statement; and (iii) for each occurrence updated, the old occurrence value. A special requirement on the log for deferring SIC checking until the end of transaction is the following. If the SIC contains *new/old* functions, the old values of the objects should be logged at the beginning of the transaction even if the values are not referenced by any transaction statement. For example, we may have a SIC to restrict $old(sum(Employee.Salary)) \leq new(sum(Employee.Salary))$, but a transaction does not reference or operate on *"sum(Employee.Salary)"*.

If the violation action of the related SIC is "warning", the intended access operation is allowed, but a warning message would be issued. If the violation action is "propagate", the access operation would also be allowed, but a related corrective action would be performed (some message may also be issued). If the violation action is "reject", the access operation would be rejected, an error code would be raised, and some explanation message describing the occurrences in violation of the SICs would also be given. In this latter case, if the transaction has proper exception-handling procedures, this error would be handled as planned (e.g., change the operation in some way and resubmit it, or skip the operation). Otherwise, the integrity maintenance subsystem would cooperate with the recovery mechanism to rollback the transaction automatically.

The SIC Representation Model SIC representation in terms of the Representation model would facilitate their enforcement since the necessary operational semantics are included. The checking is usually limited to the occurrence (that is currently manipulated by the operation in the operation component) of the object type (in the object component). The database state that makes the checking relevant is unambiguously described (in the precondition component). The necessary corrective action is also clearly specified (in the violation action component). The certainty factor component serves as the ordering factor when checking a group of SICs in the same "family". Other implementation mechanisms (e.g., keeping redundant minimum or maximum data value) can be used to improve efficiency further.

SIC Maintenance After a database is populated, there is a maintenance problem — a new SIC may be added or an old SIC may be removed. Suppose that no time-stamped past generation of data is kept. In principle, deleting a SIC does not affect the current

database state. Inserting a SIC can affect the current database state only if the SIC is relevant to some object(s) on insertion. If the organization decides the new SIC will not be applicable to the existing data, it should be explicitly stated in the SIC[33]. In that case, even if the new SIC is asserted on "insertion", the current database state need not be checked. Otherwise, a complicated procedure is needed to simulate insertions of all occurrences of the related object (asserted by the new SIC) and settle possible violations. It may be easier to maintain SICs represented in the Representation model since their operation type components are explicitly expressed.

4.3.2 Other Aspects of Database Management

The introduction of certainty factors in the Representation model brings some advantages to other aspects of database management. The first is to support flexible management. The second is to facilitate intelligent query processing.

Flexible Management By allowing uncertain SICs, we would decrease the serious rigidity of SICs criticized by Brodiga [1985]. Although those SICs are uncertain, their violation might still convey some information. Usually warning messages are issued. The purpose of issuing warning messages is twofold. First, specific occurrences that violate an uncertain SIC deserve further examination either interactively or in batch to assure that the real organizational situation is reflected. Note that the enforcement of a good uncertain SIC will generate warning messages for those, likely erroneous, (but comparatively a few) cases without interfering with the processing of routine cases. This corresponds to the idea of exception reporting. Second, if warning messages are often

[33]For example, if a new restriction on employee's salary is published on "1991/2/1" and is only applicable to new employees, it should be attached the precondition: *Employee.Hiredate* $\geq$ "1991/2/1".

issued for a SIC[34], the organization might need to reevaluate its organizational policy, which might invoke SIC maintenance.

Knowledge-based Query Optimization and Deductive Capabilities One advantage of introducing certainty factors in the Representation model is to facilitate query processing. An uncertain SIC is similar to a heuristic for directing query processing to the mostly likely objects first in an attempt to reduce response time. Furthermore, uncertain SICs can be applied to provide deductive capabilities under uncertainty. For example, from some sources we have the input data as *"with 90% certainty, a ship is a tanker, i.e., Ship.Type=tanker"*. If we have a SIC *"with 80% certainty factor, a tanker carries oil (i.e., if Ship.Type=tanker then Ship.Cargo=oil)"*, we can conclude that with 72% (i.e., 90% × 80%) certainty, the *Ship.Cargo* may be *oil*. Thus, this approach allows the Representation model to apply to expert database systems (i.e., the integration of knowledge based systems and database systems) as well as traditional database systems.

[34]For a SIC involving aggregate functions, it is possible that even if one warning message is issued only once, the organizational policy needs to be re-evaluated because violating such a SIC might imply that a number of occurrence exceptions have happened so far owing to the nature of aggregate functions. For example, there might be two SICs such as *SIC-4: with 100% certainty, avg(Employee.Salary)* $\leq$ *\$25,000* and *SIC-5: with 90% certainty, avg(Employee.Salary)* $\leq$ *\$20,000*, but no other SICs on *Employee.Salary*. Then, a single violation for SIC-5 might imply that many employees have too high salary, the salary policy may need to be reevaluated.

Chapter 5

An Extended E-R Model Incorporating Semantic Integrity Constraints

This chapter proposes a conceptual modelling tool, called **the E-R-SIC model**, for incorporating SICs. Section 5.1 describes the shortcomings of previous E-R models for dealing with SICs. Section 5.2 introduces the primitive constructs and data abstractions of the E-R-SIC model and describes the basic properties of SICs. Sections 5.3 to 5.6 explore SIC properties in more detail. Section 5.7 summarizes the E-R-SIC model in some figures.

5.1 Problems with Previous E-R Models

Existing database design methodologies based on the E-R model provide little support for incorporating SICs. Rather, they are treated as unessential accessories to conceptual modelling.

The E-R model was originally proposed by Chen [1976]. It has three primitive constructs: entity, relationship and attribute. More recently, researchers (e.g., [Smith and Smith, 1977a], [Smith and Smith, 1977b], [Teorey et al., 1986]) have extended Chen's original E-R model to provide some powerful data abstractions, e.g., generalization, aggregation, association, etc. Very few SICs are discussed in the Chen's original E-R model:

- An *incidence constraint* requires that the existence of a relationship occurrence always depends on the existence of the participating entity occurrences ([Furtado, et al., 1988]).
- The existence of an occurrence of a *weak entity type* depends on some occurrence(s) of another entity type ([Chen, 1985]).
- The maximum cardinality specifies the maximum number of occurrences of a relationship that can be related to one occurrence of an entity type.
- Attributes cannot exist on their own, but are always attached to entities or relationships.
- Primary keys must have unique values.

The extended E-R models proposed by previous researchers introduce a few more SICs, e.g., minimum cardinality. However, in all cases, SICs are only considered as accessory properties of relationship or entity types (usually a single one). For example, researchers (e.g., Palmer[1978], Tsichritzis and Lochovsky [1982]) usually discuss the optionality property of a relationship type (i.e., a relationship type, as a mapping, is total or optional to an entity type). One problem with this modelling approach is that SICs do not receive adequate attention in the conceptual design phase and are usually lost during the transformation process from the E-R model to the relational model. SICs are not treated as essential to conceptual modelling. Instead, they are considered mainly for selecting the appropriate logical data structure (e.g., relations). Because some semantics can not be considered as "properties" of a single entity or relationship type, the traditional approach may not identify all necessary SICs. For example, consider the E-R diagram where an entity type E is related to entity types F and G via relationship types RX and

RY, respectively. What SICs (if any) are implied by the adjacency of these relationship types in an E-R diagram? There may be a SIC to require that an *RX* occurrence relating to an *E* occurrence must exist if the corresponding *RY* occurrence exists, or conversely. Alternatively, there may be a SIC to require that *RX* be exclusive to *RY*. Such kinds of SICs are seldom discussed. Since SICs express restrictions on the logical meaning of data, it is worth considering the SIC as an distinct modelling construct.

5.2 An Overview of the E-R-SIC Model

The E-R-SIC model is an extended E-R model. It can be used for incorporating SICs in a database schema.

5.2.1 Primitive Modelling Constructs

The proposed model is application domain-independent. There are four primitive modelling constructs in the E-R-SIC model — entity, relationship, attribute and SIC. These four constructs are all needed to model data semantics.

Construct Description These primitive constructs are defined as follows:

Definition 5.1 *An* **entity** *is the database representation of a real-world object that can be distinctly identified.*

Definition 5.2 *A* **relationship** *is the database representation of an association among real-world objects.*

Definition 5.3 *An* **attribute** *is the database representation of a property of a real-world object or an association that is a function mapping an entity type or a relationship type to values.*

Definition 5.4 *A* **SIC** *is a logical, invariant restriction on the static state of a database (that is, a collection of attributes, entities and relationships), or on the database state transition caused by an insertion, deletion, or update operation.*

Entity, relationship, and SIC occurrences are classified into different **types** according to some criteria. At a particular moment, certain groups of entity, relationship, or SIC occurrences can be considered as **sets** in the mathematical sense, and these sets may have some aggregate properties.

Following the arguments on page 59, the E-R-SIC model does not allow the "*not applicable null*". A database designer should properly define entity or relationship types to avoid that problem.

A relationship type cannot be a participant in another relationship type. If a relationship type participates in another relationship type, it will be modelled as an entity type and the original E-R diagram will be changed appropriately[35].

Since neither relationships nor attributes can exist without entities, and a relationship can be modelled as an entity, the following is clear:

[35] That is, if a relationship type RX, which has two entity type participants E and F, is also a participant in another relationship type RY, then a new entity type, say G will be used to replace the original relationship type RX; and two new relationship types, say RE and RF, will be created to connect this new entity type, G, with entity types E and F, respectively. The new entity type, G, becomes a participant in the relationship type RY.

Entity is the most fundamental construct among attribute, entity, and relationship.

Weak Entity Type The existence of an occurrence of a *weak entity type* depends on the existence of some occurrence(s) of another entity type — the "regular" entity type. An *existence subset* [Webre, 1983] is the subset of occurrences of the regular entity type upon which a weak entity occurrence is dependent. The E-R-SIC model requires that the above relationship be explicitly specified in order to make clear the "dependence semantics" — the existence subset for each weak entity occurrence[36].

Time The E-R-SIC model does not model "time" as an entity type[37]. That is, time-stamped past generations of data are not modelled. However, there may be SICs applied to some time-valued attributes, temporal sequences between object occurrences, or the special system variable, *Current_time*.

Simplifying Assumptions For simplification, there are two assumptions on relationships.

[36]In the literature there is a related discussion on another term, weak relationship. However, note that researchers use that term to imply diverse meanings. For example, Tsichritzis and Lochovsky [1982] use it to mean the relationship between a weak entity type and its related regular entity type. Scheuermann et al. [1980] classify **weak relationships** into two types. Their second type should be modelled by an association abstraction. Their first type (also see [Dogac and Chen, 1983]) implies a special SIC, which is called **Critical_Relationship_Occurrence SIC** in this research. That is, the existence of an occurrence of an entity type E depends upon the existence of exactly one critical occurrence of a relationship type R, via which it related to another entity type F. An occurrence of E cannot exist if its related critical relationship occurrence of R does not exist though it may still participate in other occurrences of R. For example, *each employee should be assigned to at least two projects, and exactly one of these would be critical.* In order to model such a situation, the relationship type R should have a special attribute, e.g., "*Criticality*", which is a binary variable to indicate whether a relationship occurrence is critical.

[37]If we model "time" as an entity type, all other entities and relationships would connect with "time" via at least two special relationship types — "has_creation time" and "has_deletion time" ([Studer, 1988]). The E-R diagram would become very complicated.

No Ternary or Higher Degree of Relationships Ternary relationships and relationships of higher degree are not discussed in this dissertation because of their additional complexity[38]. Instead, ternary relationships or relationships of degree greater than two are simulated using binary relationships ([Kent, 1977])[39].

No Recursive Relationships The relationships discussed in this dissertation are binary relationships involving two distinct entity types. Recursive relationships, which are relationships involving only one entity type, are simulated by binary relationships. That is, one or two[40] subtypes will be created for the entity type participating in a recursive relationship type. Such a simulation avoids the need to attach an explicit "role" when referencing an entity type participating in a relationship type. These subtypes also make the semantics clearer in most cases[41]. When an entity type is required to participate in both roles of a recursive relationship, such a distinct subtype, although it may be redundant, will be helpful for understanding the semantics. We may have some problems when an entity type is required to participate in both roles of a recursive relationship

[38]For example, the cardinality specifications are more complicated. In a ternary relationship, there are twelve pairs of minimum and maximum cardinalities according to the cardinality definition given by Lenzerini and Santucci [1983].

[39]That is, in the case of ternary relationships among entity types, the first binary relationship is formed between two entity types, the second binary relationship is then formed to connect the first binary relationship with the remaining entity type. Since the participants of a relationship type should be entity types, the above means that the first binary relationship should become an entity type. For example, suppose that there is a ternary relationship *Order* among three entity types, *Parts, Warehouses,* and *Suppliers.* We will model the situation as four entity types, *Parts, Warehouses, Allocation, Suppliers*, and three binary relationship types connecting *Allocation* with *Parts, Warehouses, Suppliers*, respectively. *Allocation* is a new entity type converted from the original binary relationship between *Parts* and *Warehouses.*

[40]One subtype is sufficient to replace a recursive relationship with a binary relationship although a database designer may prefer to have both. For example, in the relationship type *Supervise*, *"Employee (0,*) Supervise Employee (0,*)"* where * stands for some positive number other than 0, one subtype (e.g., *Supervisor*) is sufficient.

[41]Compare to an extreme case where there is only one entity type "*Thing*" in the database and all relationships are recursive.

and those two roles are not distinct[42]. However, such cases are very idiosyncratic. For simplification, this research is limited to handling binary relationships with two distinct entity types.

5.2.2 Data Abstractions

A data abstraction is a simplified description of a system that suppresses specific details while emphasizing those pertinent to the problem. Like other extended E-R models in the literature, the E-R-SIC model provides three kinds of data abstractions: inclusion, aggregation, and association.

Inclusion Abstraction

The **inclusion** abstraction concept ([Goldstein and Storey, 1990]) encompasses **classification**, **generalization**, and **specialization**. Classification is a form of abstraction in which a type is defined as a collection of occurrences with common properties. Specialization occurs when every occurrence of a type is also an occurrence of another type. Specialization is indicated by the term, *is_a*, that is, S *is_a* G, where S is a subtype and G is a super-type. It is possible to have generalization or subset hierarchies ([Teorey, et al., 1986]). A generalization hierarchy occurs when a type is union of non-overlapping subtypes. A subset hierarchy occurs when a type is union of possibly overlapping subtypes. Property inheritance, which means that attributes, associated relationships and imposed SICs of the super-type are inherited by each subtype (or a type's are inherited by its occurrences), is the most important characteristic of the inclusion abstraction. In the

[42]That is the relationship is symmetric, e.g., *"Person Friendship Person"*

case of the classification abstraction (related occurrences to types), the property inheritance principle has been enforced traditionally by any DBMS. This research assumes that the DBMS (either in the E-R model or the relational model) would also automatically implement property inheritance for the specialization abstraction. That is, the principle of redundancy minimization, i.e., properties that can be inherited from some other entity type via an *is_a* relationship should not be stored explicitly ([Goldstein and Storey, 1990]), has been followed. Otherwise, the inherited attributes, relationships and imposed SICs would need to be explicitly and redundantly stored for each specific subtype and other SICs would be required to assure that these properties have been really "inherited"! However, there is one exception to the use of redundancy minimization principle. Although the primary key of the super-type may not be chosen as the primary key of the subtype, it is indeed an attribute and candidate key of the subtype. In order to make the semantics of a subtype entity clearer and have the same related SIC(s) regardless of the choice of the primary key, the primary key of the super-type is suggested to be stored in the subtype.

Aggregation Abstraction

Aggregation is an abstraction that allows a relationship between objects (i.e., attributes, entities, relationships) to be considered as a higher level object ([Smith and Smith, 1977b]). The term, *component_of*, is used to indicate the aggregation, that is, *C component_of A*, where *C* is a component and *A* is an aggregate. The E-R-SIC model provides **composite entity aggregation** that is an abstraction in which an entity contains other dependent entities and some attributes as real components[43]. The aggregate entity

[43]In addition to composite entity aggregation, aggregation abstractions can be classified into four other kinds based on discussions about aggregation in the literature. (1) *Attribute aggregation* is the abstraction in which an attribute may be defined as the aggregation of attributes. This research does

"owns" these other entities ([Lee and Lee, 1990]). That is, the existence of these other entities is dependent on their aggregate and are owned by exactly one aggregate (i.e., the cardinalities of any component in the *component_of* relationship are always (1,1)). Although some researchers argue that in aggregation the inheritance is upwards ([Brodie, 1983, p. 576], [Mees and Put, 1987], [Potter and Kerschberg, 1988]), this research argues that it is probably more suitable to state that the aggregate "owns" components and components "own" their attributes, so the aggregate "owns", rather than "inherits", components' attributes. For example, we may state "*a car owns engine.weight, engine.brand, and brake.brand, etc.*".

Association Abstraction

Association is the abstraction in which a collection of member objects is considered as a higher level (more abstract) set object ([Brodie, 1983]). The term *member_of* is used to indicate the association, that is, *M member_of S*, where *M* is a member and *S* is a set. Brodie [1983] states "as with aggregation, the inheritance goes upward" and some researchers (e.g., Mees and Put [1987]) even state that association may support both upward and downward inheritance. This research takes the position that there is no inheritance in association. That is, set is distinguished from type and a "set" in association is considered to represent a pure mathematical set.

---

not consider the hierarchical structure of attributes. (2) *Simple entity aggregation* is the abstraction in which an entity is defined by aggregation of attributes. This is the traditional entity concept. (3) *Complex entity aggregation* is the abstraction in which an entity is defined by aggregation of attributes, other independent entities, and probably sets. The aggregate object does not really "own" them as components. That is, the cardinalities of these "components" in the *component_of* relationship may not be (1,1). This aggregation may convey less semantics because these "relationships" between the aggregate object and its "components" are all implicitly named as "component-of". Therefore, the E-R-SIC model does not adopt it. (4) *Relationship aggregation* is the abstraction in which a "relationship" is obtained by aggregation of entities and some attributes. It is just another way to represent a relationship.

Before Brodie mentioned association, dos Santos et al. [1980] had already proposed a useful data abstraction "**correspondence**", which was later referred to by Furtado and Neuhold [1986] as "**grouping**". Grouping is more general than association. It creates a group of sets, i.e., grouping is an abstraction that defines a new entity type in which each occurrence is a set formed from a collection of occurrences of the source entity type.

According to the correspondence idea in [dos Santos et al.,1980], a group of sets is formed by an *indexing* mechanism. Applying the idea of correspondence, the E-R-SIC model provides three kinds of associations as below.

1. **Natural Set Association:** A set, S, is defined to contain all occurrences of an entity M type. Classification is the indexing mechanism to form a set. For example, we have: *"Employee member_of Employees"* where *Employees* is a set consisting of all *Employee* occurrences in the employee type; or *"Department member_of Departments"*.

   If these sets only have attributes that are derived from those of their members, their explicit representations may be redundant and may not be efficient after considering the enforcement of SICs. However, there may be *a priori* attributes, for example, *Employees.Representative.* In the whole database, there are a number of such sets, e.g., *Employees, Departments.* Since they may have different derived attributes and *a priori* attributes, they form different entity set types containing a single occurrence, respectively. This association abstraction relationship, *member_of*, should not have its own attributes and need not be explicitly represented.

2. **Indexing Derived Set Association:** This is the original correspondence abstraction discussed by dos Santos et al. [1980]. The indexing mechanism[44] can be an

[44]Formally, Furtado and Neuhold [1986] define grouping as below. "If T designates some entity set

indexing attribute that is an attribute of the indexed entity type, or an indexing entity type that is related to the indexed entity type via an indexing relationship type. An example is *cosets of employees of the same age*, where *Employee.Age* is the indexing attribute for the indexed entity type *Employee*. If the indexing attribute is an attribute that disallows null ("unknown"), the cosets obtained by grouping would form a partition of the indexed entity type occurrences. Another example is shown in Figure 5.1, where *DS* is the indexing derived set (e.g., *cosets of employees who work_for the same project*), *M* (e.g., *Employee*) is the indexed entity type, *I* (e.g., *Project*) is the indexing entity type, and *R* (e.g., *Work_for*) is the indexing relationship. Grouping is a powerful abstraction. There may be more than one indexing type, which can be combined with indexing attributes as the indexing mechanism. Although we can get a group of sets from the indexing mechanism, we may only be interested in some of these (e.g., *the set of employees who work_for project p100*).

In general, this association abstraction relationship, *member_of* does not have its own attributes and need not be explicitly represented. This kind of set has some attributes derived from the indexed entity type, some are defined *a priori*. Two kinds of attributes in a set are important for membership derivation: the indexing attribute (e.g., *Employee.Age*) and the primary key of the indexing entity type (e.g., *Project.Id*).

3. **Enumerated Set Association:** There is no indexing mechanism in this kind of association because the database designer does not know or does not care about

---

and $T_1, T_2, ..., T_n$ are either value sets associated with T or entity sets related via some relationship with T, then the Grouping operator denoted by $T_1, T_2, ..., T_n$ {T} constructs a new (grouped) entity set T_G where each element is a set of entities of T such that inside of one such set all entities have the same values and related entities from the entity sets $T_1, T_2, ..., T_n$ associated. The types $T_1, T_2, ..., T_n$ will be called *indexing types*, T the *basis*."

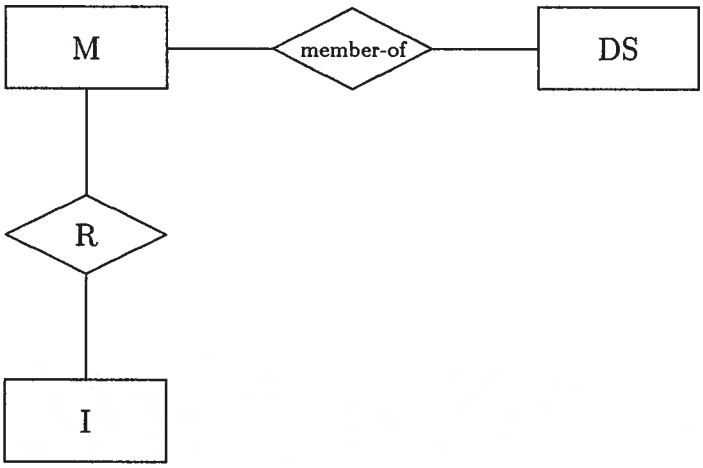


Figure 5.1: Grouping: Member (M), Derived Set (DS), Indexing Entity (I), Indexing Relationship (R)

the indexing entity types or attributes. The set membership can only be explicitly enumerated by the *member_of* relationship.

5.2.3 Basic Properties of SICs

Based upon the ontological concepts of Bunge's formalism, Wand and Weber ([1988]; [1989]; [1990]) develop a formal model of the deep structure of an information system. Their work begins with a fundamental premise that is an adaptation of Newell and Simon's [1976] physical symbol system hypothesis. Although as described in Section 1.3 the SICs in this research are only parts of their "laws", we can borrow this premise to develop the SIC concept.

Premise 5.1 *A physical-symbol system has the necessary and sufficient properties to represent real-world meaning.*

An information system is a physical-symbol system. A database is part of an information system and consists of attribute, entity and relationship occurrences. In order to represent data integrity semantics, there may be constraints restricting the existence or change of these occurrences. Some constraints may specify necessary conditions that must hold for an occurrence to exist, not exist or change in a database. Other constraints specify sufficient conditions, which if true, imply that an occurrence must exist, not exist, or change in a database. Thus, we can interpret a constraint as follows.

> *A SIC is an assertion — a sufficient or necessary condition — for an occurrence of an attribute, entity, or relationship type to exist, not exist, or change in a database.*

What are these conditions?

- Although these conditions are usually specified for an entity or relationship type or its attributes, they are actually restrictions on the insertion, deletion, or update of its occurrences in a database, rather than on the addition or removal of the type from the database schema.
- A SIC is defined intensionally in a database schema rather than extensionally. The conditions apply to an occurrence by virtue of the fact it belongs to an entity, relationship, or attribute type. However, the conditions for entities or relationships may only be relevant to some occurrences of the specified type. These occurrences, indeed, can be considered to belong to an "implicit subtype".

Reasons of having implicit subtypes. By applying the classification abstraction ([Schrefl et al., 1984]) those entity/relationship occurrences with common properties form an entity/relationship type; and by the principle of property inheritance every entity/relationship occurrence should have exactly all the attributes of the entity/relationship type and conform to the SICs associated with the entity/relationship type ([Knuth et al., 1988]). However, these occurrences may not be "homogenous". That is, there may be special SICs on some occurrences. One could create a subtype for only those special occurrences. By taking this approach (similar to [Dampney, 1988]) we could avoid a number of special types of SICs associated with "implicit subtypes". However, since there may be a number of such special SICs, there would have to be a corresponding number of such subtypes in a database. The organization may have no intrinsic interest in these explicit "subtypes". In this research, the subtypes will be created only if they are meaningful to the organization or if they are needed to eliminate recursive relationships. That is, we have the following premise:

Premise 5.2 *The occurrences of each entity or relationship type are not homogenous; that is, they have different attribute values, and are related to different occurrences of other objects.*

This implies that a database designer could not specify all entity and relationship types such that all occurrences of each type satisfied the same set of SICs. That is, "implicit subtypes" with unique SICs always exist.

An entity subtype may be implicitly defined by restricting attribute values of its occurrences or by requiring occurrences to participate in some relationship types[45].

[45] In another semantic data model, SDM, there are four ways to define "sub-classes": attribute-defined, user-specified, set operator-defined, and existence-defined ([Hammer and McLeod, 1981], [Urban and Delcambre, 1986]). A user can decide what occurrences the subtype will contain. From the SIC perspective,

A relationship subtype may by implicitly defined by restricting attribute values of its occurrences, by requiring its participating entity occurrences to participate in some other relationship types, or by restricting attribute values of its participating entity occurrences. A condition for an entity or relationship type can be considered as the definition of an implicit subtype if there are other conditions and this condition provides a criterion to decide whether an occurrence satisfies other conditions (that are special SICs for the "implicit subtype").

- These conditions may be positive or negative. A positive condition requires that something must happen. A negative condition requires that something must not happen. For attributes, by simply reversing the comparison operator (e.g., reversing *"not E.A> value"* to "E.A $\leq$ value"), we could consider only positive conditions. However, we must consider both positive (e.g., *participate_in*) and negative (e.g., *not participate_in*) conditions for relationships.

- These conditions may be simple assertions, or complicated arithmetic formulas. To be considered a single SIC, a condition should be "atomic", i.e., indivisible. Otherwise, it is really two or more SICs. That is, usually we need not consider conjunctions of conditions. However, there may be some "further restrictions" that are conditions on some basic restrictions. For example, there may be a SIC such as, "*if the salary of an employee is greater than $40,000, he (she) must participate in some project, which is project1*", in which the last part of the statement is a further restriction on the participation in a *project*.

- These conditions may only require that any one occurrence of a specific type exist, or more strictly, may require that *at least, at most, or some exact number* of

---

the semantics are embedded in the user's mind and it is left to the user to maintain the database in order to reflect real-world changes. If a subtype is defined by some set operation, the definition is explicit. The ideas of the "attribute-defined" and "existence-defined" "sub-classes" in SDM are adopted here.

occurrences of the specific type exist. That is, the restriction may be **qualitative** or **quantitative**. Since an attribute type has exactly one occurrence per entity or relationship occurrence, no quantitative requirement is placed on attributes. The quantitative requirements are more important for a relationship type. They are the relative cardinalities of a relationship type — restrictions on the minimum, maximum, or exact number of relationship occurrences in which an occurrence of the specified entity type participates; or requirements that for each occurrence of the specified entity type, all occurrences of the other entity type relate to it via the relationship type.

- These conditions may directly restrict the value of an attribute of a single entity or relationship occurrence, or may restrict the aggregate value of an attribute for a set of entity or relationship occurrences. All occurrences of an entity or relationship type (or its "implicit" subtype) naturally form a set (although the database designer may not explicitly define it). The most important set properties are the *counting number, minimum, maximum, summing and average values.*

- Although a condition may be asserted for occurrences of **one** entity or relationship type, it might also be for occurrences of **a group of** types taken together.

- These conditions may explicitly involve **time** to restrict or trigger the existence, nonexistence, or change of an occurrence, or may assert a temporal sequence between the existence of occurrences[46].

[46] Without time-stamped past data, the temporal requirement would imply "no time lag" between the existence of adjacent occurrences in a sequence of objects.

Systematic Modelling A naive approach to modelling SICs would enumerate all necessary and sufficient conditions for the existence, nonexistence and change of the occurrences of each attribute, entity, and relationship type. However, if a SIC involves several objects, we need not specify it several times. For example, suppose that a SIC requires that if an *RX* occurrence exists for an occurrence of the specified entity occurrence, an *RY* occurrence must also exist. We would specify this SIC when we identify necessary conditions for the existence of *RX*. Although the existence of *RX* is also a sufficient condition for the existence of *RY*, it need not be incorporated again as a general constraint if we properly represent the above SIC (i.e., by decomposing it into two sub-SICs in terms of the Representation model). It would be desirable to be able to guarantee that all conditions on all objects have been completely covered although we need not explicitly consider all kinds of conditions for each object. The systematic modelling procedure proposed in the remainder of this chapter and the decomposition algorithms proposed in Chapter 7 are used to achieve this goal. By applying the procedure described in Sections 5.3 to 5.6 a database designer can model SICs systematically by first considering the necessary and sufficient conditions for the existence, nonexistence, and change of each attribute of each entity and each whole entity in isolation, and then examining the whole E-R diagram by considering each explicit or implicit relationship.

Representation Languages Although SICs incorporated by the E-R-SIC model can directly be represented in terms of the SIC Representation model, it is suggested that they are first represented in a simplified format using rules or expressions. The purpose of using this simplified format language is to represent the precondition and predicate components first in a language that is closer to natural language. The relevant objects and operation types can be derived from the simplified format by applying the algorithms

introduced in Chapter 7. The certainty factor and violation action specifications are closely related to the objects and operation types, and can be added later.

The simplified format contains the preconditions, predicates, and some operation type information (by the keywords, *is_to_be_deleted*, and *is_to_be_updated* for operation-dependent SICs), but not the certainty factor and violation action. For example, we may represent a requirement on the salary of any programmer first in the simplified format as:

if Employee.Title="programmer"
then Employee.Salary≥18000.

BNF descriptions of the simplified format are described in Appendix C. For convenience, this dissertation describes a SIC as being in rule format if it can be written by using the keywords *if* and *then* (and the *"if"* condition part is not empty) in the simplified format. Since a relationship type name usually can be a verb phrase, when a relationship type is discussed, its participating entity types will be mentioned too. Because a relationship type involves two entity types, we need a *with_respect_to* keyword to clarify to which entity type we are referring[47]. An assertion may have a temporal modifier (*before* or *previously*) in a rule describing a temporal sequence requirement between the existence of the object in this assertion and the existence of the object in another assertion. If a *"before"* modifier is attached to an assertion, the assertion must be true for its involved object at the time another assertion is becoming true[48]. If a *"previously"* modifier is attached to an assertion, at the time the assertion becomes false for its involved object,

[47]For example, *"if with_respect_to E, E RX F then with_respect_to E, ¬(E RY F)"* is a SIC to assert that *RX* and *RY* are mutually exclusive relative to *E* only.

[48]For example, we may have: *"if Dealer Own Car then Manufacturer Possess Car before"*.

another assertion must become true[49].

SICs written in the simplified format should be later reformulated in terms of the SIC Representation model by applying the algorithms introduced in Chapter 7. Most of designer-specified SICs are general SICs. These general SICs must then be decomposed into several operation-dependent sub-SICs in terms of the Representation model. For example, the static constraint in the simplified format shown on the preceding page will be represented as three operation-dependent SICs — one each on update of *Salary*, update of *Title*, and insertion of *Employee*. At this time, the certainty factor and violation action specifications are added.

SIC Types The E-R-SIC model can be used to classify SICs into a number of domain-independent SIC types. This SIC type classification facilitates development of SIC consistency and nonredundancy rules that can be applied to make SIC elicitation more efficient. It also allows us to present generic SICs for some common SIC types so that the number of SICs that must be specified using the Representation model can be greatly reduced. Appendix D provides a detailed description of this classification.

5.3 Entity Attribute SICs

First examine necessary and sufficient conditions for the existence/nonexistence/change of each attribute of each entity in isolation.

Necessary Conditions An attribute cannot become nonexistent once it exists. Therefore, we only need to consider necessary conditions for its existence and change.

[49]For example, we may have: *"if Manufacturer Possess Car previously then Dealer Own Car"*.

Associated Entity Type SICs If an attribute exists, its value may be restricted to be in a data type and in some range, and/or may be specified as not-null that means the *"unknown"* value is not allowed. In addition, its value must be expressed in some specified format that is meaningful to the organization. If an attribute is allowed to be updated, it must be specified as changeable. All attributes (including primary keys) of relationships or entities are updatable unless declared otherwise. If it can be updated, there may be special SICs restricting the pairs of before and after values.

Associated Entity Set SICs Because an attribute type is defined as a function from an entity type to value, it has only one occurrence per entity occurrence, i.e., it is single-valued, not multi-valued. We need not consider the attribute set. However, because an attribute belongs to an entity, there may be some SICs restricting entity set properties. They may require that each attribute value be unique. The restrictions on minimum, maximum, summing, or average value of the attribute in an entity set may also be specified.

Primary Key Problem In the E-R-SIC model, following the traditional E-R model and relational model, there is no internal identifier (*"surrogate"*) to represent an entity or relationship. Rather, some attribute or combination of attributes is used as the primary key. Unfortunately, this overloading causes the semantics implied by an update of a primary key to be ambiguous — it may imply a simple update of an attribute or it may imply the deletion of an "old" entity (relationship or tuple) followed by an insertion of a "new" one. The update of a primary key is allowed in the E-R-SIC model. However, the SICs related to deletion and insertion must be enforced.

Time Restriction There may be some SICs restricting a time-valued attribute. The expression of such SICs may involve an explicit *Current_time* variable[50].

Sufficient Conditions An attribute type is included by a database designer to satisfy organizational needs. Because the "*not applicable null*" is not allowed, for each entity occurrence, there is at least one attribute occurrence. Therefore, no SICs can be expressed as sufficient conditions for the existence or nonexistence of an attribute considered in isolation.

If we consider a single attribute in isolation, neither is there any sufficient condition for attribute change except time-triggering. That is, an increment of the *Current_time* might trigger an update to an attribute. For example, *at 0:00 on 1/1/1993, increase the salary of each employee by $1,000.*

5.4 Entity SICs

Examine necessary and sufficient conditions for the existence/nonexistence/change of each whole entity type in isolation.

Necessary Condition

Entity Type SICs If an entity occurrence exists, there may be a formula[51] among its attributes. In general, any entity occurrence can be deleted to reflect the real world

[50]For example, *Current_time* $\geq$ *(Employee.FirstWorkDate + "2 years")* is an expression to assert a rule that *"each employee must have at least 2 years working experience"*.

[51]If there is more than one attribute appearing in an expression, we call that expression a formula.

situation. However, there may be SICs that require some of its attributes to have certain values before the deletion may take place.

Entity Set SICs Because an entity occurrence belongs to an entity set, there may be SICs restricting set properties. They may specify the maximum number of occurrences of an entity type that are allowed to exist in the database. The concatenated values of some attributes may be required to be unique. Some aggregate values of attributes may be required to be interdependent[52], or more strictly, satisfy a formula. There is no restriction on the minimum number of occurrences that can exist in the database because in the beginning there are no occurrences, and we should allow that an entity type may have no occurrences temporarily even after the database is already populated.

There are some further complicating factors such as the following:

The First Complicating Factor — Implicit Subtype The single attribute SICs in Section 5.3 and the SICs mentioned above in this section may be conditional. That is, they may include conditions that can be thought of as defining "implicit subtypes". If we consider a single entity type in isolation, the only way to define "implicit subtypes" is by some range restrictions or formulas involving its attribute values. It is possible that a SIC (e.g., nonvolatility, value restrictions, etc.) applies to a single attribute only for a specified "implicit entity subtype". It is also possible that a SIC (e.g., restriction on the maximum number of occurrences, etc.) applies to an occurrence only because this occurrence belongs to a specified "implicit entity subtype".

specifications

[52]For example, *if min(E.A1) > v1 then avg(E.A2) ≥ v2.*

The Second Complicating Factor — Time Restriction There may be SICs that restrict other attributes when time-valued attributes satisfy certain conditions. As time passes, the restriction will either finally disappear or come into force[53]. These explicit time restrictions may be added to all SICs mentioned in this section and in Section 5.3. It is possible to combine time restrictions and "implicit subtypes" factors in a single SIC.

The Third Complicating Factor — Temporal Sequence among Attributes There may be some temporal conditions among the attributes of an entity. They may require that an entity occurrence's attributes must satisfy a certain condition at the time one of its other attributes is going to acquire some value[54]. It is also possible to require that if its other attribute(s) satisfied a certain condition in the past (and no longer satisfies it now), one of its attribute must take some value.

Sufficient Conditions An entity type exists in the database only because the database designer includes it in order to satisfy the interests of the organization. It would be very unusual to have a SIC requiring a specific entity occurrence to exist or not to exist in a database. Therefore, we exclude the possibility of expressing a SIC as a sufficient condition for the existence or nonexistence of an entity occurrence considered in isolation.

If we consider a single entity type in isolation, neither is there any sufficient condition for entity change except time-triggering. That is, for an entity occurrence, an increment of the *Current_time* might trigger an update to one of its attributes if its other attributes

[53]For example, we may have a SIC like "*an employee cannot receive a salary raise during his(her) first 6 months in the company*", or *"if an employee has worked for at least two years, he (she) must have at least 10 vacation days"*.

[54]For example, we may have a SIC: "*if E.A1=v1 then E.A2=v2 before*".

satisfy some condition(s).

5.5 Relationship SICs

Now traverse an E-R diagram and consider the necessary and sufficient conditions for the existence/nonexistence/change of each relationship. In terms of topology, the important local contexts describing how relationships and entities are connected together in an E-R diagram are[55]: *line, star, loop-2,* and *loop-n.* Each relationship type is in one or more of these contexts. When incorporating SICs for a relationship, the database designer needs to recognize its contexts.

Definition 5.5 *If an entity type participates in a group of relationship types, it is a* **sharing entity type** *to them.*

Definition 5.6 *If in a part of an E-R diagram, each entity type participates in at most two relationship types and there is no cycle (loop) among these entity types, e.g., Figure 5.2, each of these relationships is in a* **line** *context.*

Definition 5.7 *If there is a sharing entity type participating in three or more relationship types and there is no loop among these entity types, e.g., Figure 5.3, each of these relationships is in a* **star** *context.*

Definition 5.8 *If there is a loop between two entity types, i.e., two or more relationships exist between two common entity types, e.g., Figure 5.4, each of these relationships is in a* **loop-2** *context.*

[55] If we allowed explicit recursive relationships, they would be in the *loop-1* context.

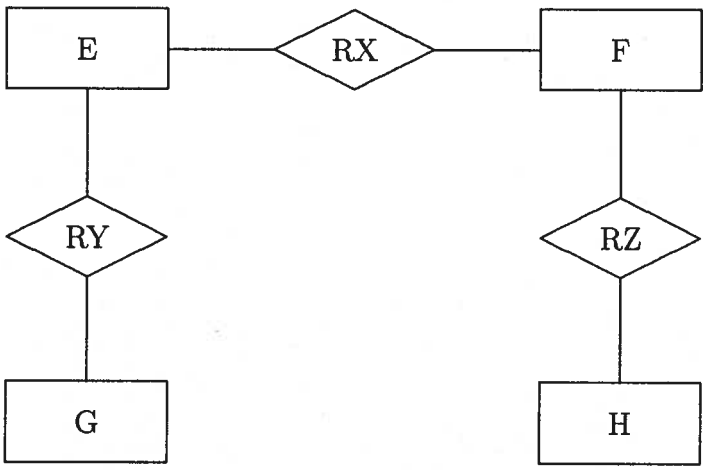


Figure 5.2: A Line Layout Context

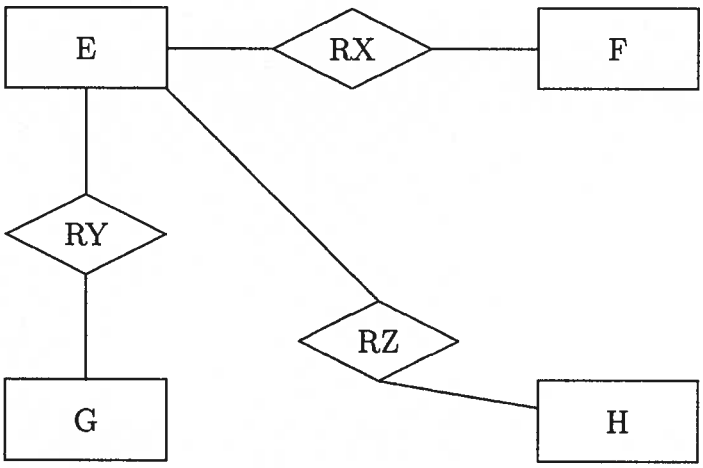


Figure 5.3: A Star Layout Context

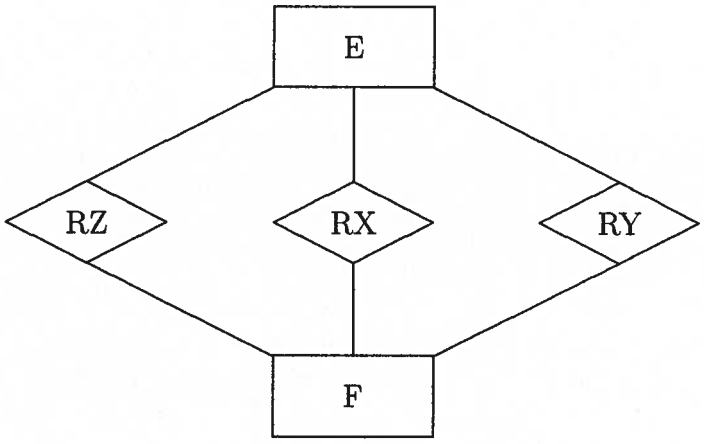


Figure 5.4: A Loop-2 Layout Context

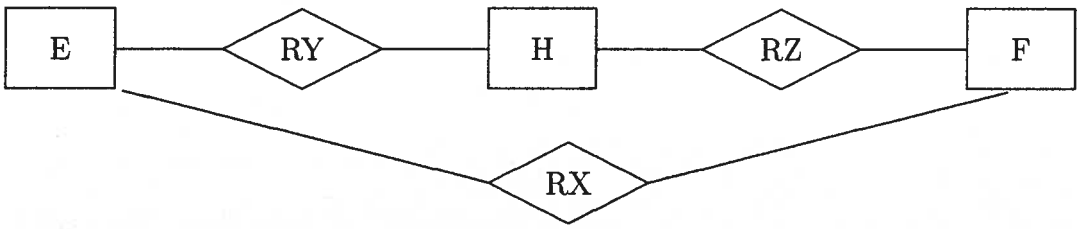


Figure 5.5: A Loop-n Layout Context

Definition 5.9 *If there is a loop among n≥3 entity types, e.g., Figure 5.5, each of these relationships forming the loop is in a* **loop-n** *context.*

5.5.1 Necessary Conditions

It is necessary to examine the necessary conditions for one or multiple occurrences of a single relationship type to exist (not exist/change), and for occurrences of a group of relationship types to co-exist.

Single Relationship Type SICs

First we focus on a single relationship type.

Focus on Attributes A relationship is similar to an entity because it can have its own attributes in addition to the primary keys from the participating entities. Therefore, a relationship can have single attribute SICs similar to those discussed in Sections 5.3 and

single relationship SICs[56] such as those of Section 5.4. Since a relationship's attributes are usually few, such SICs are not common. However, it is possible that there may be some more complicated SICs[57] because the definition of an "implicit relationship subtype" could be based on some conditions on one or both participating entity occurrences (e.g., having some attribute values, or participating in some other relationship types), or some formula involving the attributes of the relationship and its participating entities.

Focus on Association In another sense, a relationship is much different from an entity because a relationship occurrence represents the association between two entity occurrences.

Inherent SIC First, an inherent SIC is that the relationship occurrence's participating entity occurrences must exist.

Restricted-connecting Set There may be some restrictions on constructing a relationship, i.e., the necessary conditions for the existence of a relationship occurrence.

Definition 5.10 *Suppose that the relationship type* R *connects the entity types* E *with* F. *The* **restricted-connecting set** *of an entity occurrence of* E *in the relationship type* R *is those occurrences of* F *that the entity occurrence of* E *is allowed to be related to via* R.

[56] In contrast to an entity type, absolute maximum cardinality for a relationship type need not be considered although some researchers mention it. As shown by [Lenzerini and Santucci, 1983], the absolute maximum cardinality of a relationship is bounded by the absolute maximum cardinalities and relative maximum cardinalities of participating entity types.

[57] For example, there might be a SIC stating that "if Employee Work_for Project, Project_Id=p100 then Work_for.Hours $\geq$ 50".

Normally, a DBMS would not know the restricted-connecting set of each entity occurrence in each relationship since a SIC is seldom specified extensionally. However, there may be some general business rules that restrict intensionally the restricted-connecting sets. The restrictions may be positive or negative. The restrictions on the freedom to construct a relationship include:

1. **One-side condition.** The restriction may be on one entity type. It would imply that some occurrences of each participating entity type are not allowed to participate in the relationship type. That is, their restricted-connecting sets are undefined. The relationship type R is in fact defined on some "implicit entity subtype(s)" of E or F. The "implicit entity subtypes" may be defined positively, i.e., as occurrences having some specific attribute values, having attributes satisfying some formula, participating in some other relationship types; or negatively, i.e., occurrences not participating in some other relationship types. We need to consider possible disjunctions of conditions in a *star* context because there are at least two other relationship types[58].

2. **Two-side condition.** The restrictions may be coupling conditions on both entity types at the same time. If these are positive restrictions, they are stronger than the above one-side restriction. If these are negative restrictions, they are weaker than the above one-side restriction. Although all occurrences of each participating entity type, E or F, may be allowed to participate in the relationship type, they are not freely connected together. An example of a positive restriction is that each occurrence of an "implicit subtype" of E can only connect with those occurrences of some "implicit subtype" of F. An example of a negative restriction is that each occurrence of an "implicit subtype" of E cannot connect with those occurrences of some

[58] For example, in Figure 5.3, we may have: "*if E RX F then (E RY G)* $\vee$ *(E RZ H)*".

"implicit subtype" of F. The "implicit entity subtypes" on both sides of a relationship may be defined interdependently as occurrences having some specific attribute values[59], having attributes satisfying some formula, participating in some other relationship types (e.g., occurrences of E participating in RX and their connected occurrences of F participating in RY at the same time)[60]. Some SICs described in the literature are just special cases. For example, a **Subset_Relationship SIC** ([Palmer, 1978]) in a *loop-2* context[61] and the necessary conditions for the composition of relationships ([Lenzerini and Santucci, 1983], [Azar and Pitchat, 1987]) in a *loop-n* context[62], are special cases requiring that the specific occurrences of F connecting with an occurrence of E via other relationship types (e.g., RY, RZ) be in the restricted-connecting sets of the occurrence of E. An **Exclusive_Occurrence SIC** ([MaFadden and Hoffer, 1988]) in a *loop-2* context[63] is also a special case requiring that the specific occurrences of F connecting with an occurrence of E via the other relationship type (e.g., RY) not be in the restricted-connecting set of the occurrence of E.

3. **Intra-relationship condition.** It is possible that we can define a restricted-connecting set based on some properties of the relationship type. For example, we may define a restricted-connecting set of an occurrence of E by requiring that if some occurrences (e.g., having some attribute values) of F are in the restricted-connecting set, some other occurrences (e.g., having some other attribute values) must or must not be in the set. A relationship type may also be **symmetric** or **transitive**.

[59] For example, in a relationship *Drive* connecting *Driver* with *Vehicle*, even if it is total on both sides, some drivers (e.g., with class 5 driver-licence) can only drive some vehicles (i.e., with gross weight $\leq$ 10,900 kgs).

[60] For example, there may be a SIC that state: *if Employee Is_Allocated Car then if Employee Work_for Project then Car Is_Insured Collision_Insurance".*

[61] For example, in Figure 5.4, we may have: "*if E RX F then E RY F*".

[62] For example, in Figure 5.5, we may have: "*if E RX F then (E RY H)* $\wedge$ *(H RZ F)*".

[63] For example, in Figure 5.4, we may have: "*if E R F then* $\neg$*(E RX F)*".

The fundamental properties of any relationship type are assumed by default to be irreflexivity, asymmetry, and intransitivity. However, some relationships may be reflexive, symmetric, or transitive. The reflexivity of a relationship will not be discussed in this research because it is quite unusual in a traditional database. Symmetric[64] or transitive[65] relationships may occur in a specialization hierarchy, that is, when the two involved entity types belong to the same super-type; or one is a super-type (e.g., *Employee*), and the other is its subtype (e.g., *Manager*).

There are some further complicating factors such as the following:

1. **The first complicating factor — temporal sequence conditions.** A SIC may require that for a relationship occurrence of a specified type to exist, one of its participating entity occurrences must have attributes with certain values, or participate (or not participate) in other relationship type(s) at the time it is going to participate in this relationship. Those occurrences of other relationship types are allowed to be deleted after insertion of this relationship.

2. **The second complicating factor — quantitative requirements.** For a relationship occurrence to exist, there may be a quantitative restriction on the maximum number of relationship occurrences of the same relationship type, in which one involved entity occurrence has participated — that is the relative maximum cardinality to the specified entity type. If a necessary condition for the existence of a relationship involves other relationship type(s), there may be some special quantitative requirements that are cardinalities of those other relationship types[66].

[64] Some examples of such relationship types are: *Sibling_of, Married_to, Partner_of.*
[65] Some examples of such relationship types are: *Sibling_of, Ancestor_of, Supervise, Partner_of.*
[66] For example, there may be a SIC such as, *"if* $\exists$ *RX, E RX F then* $\exists$ *exactly 3 RY, E RY G".*

3. **The third complicating factor — multiple occurrences.** The above only considers necessary conditions for the existence of one relationship occurrence. However, there may be some necessary conditions for the existence of multiple relationship occurrences of the same type relative to a sharing entity occurrence[67].

Group Relationship Type SICs

The above focuses on the necessary conditions for the occurrences of a single relationship type although sometimes several relationship types may be involved in the conditions. In a *star* context, there are several relationship types with a sharing entity type. If several occurrences of different relationship types co-exist, there may be some necessary conditions to require that the sharing entity must participate in some other relationships, have some attribute values, or there may be a formula among some attributes of the sharing entity and those relationships.

Relationships Involved in Defining "Implicit Entity Subtypes"

It is necessary to review those single attribute SICs in Section 5.3 and Single Entity SICs in Section 5.4 in the presence of relationships. For example, it is possible that a SIC (e.g., nonvolatility, value restrictions, etc.) applies to a single attribute only because its associated entity participates in some relationship(s). It is also possible that a SIC (e.g., restriction on the maximum number of occurrences, etc.) applies to an entity occurrence only because the occurrence participates in some relationship(s).

[67] For example, there may be a SIC such as, *"if* $\exists$ *exactly 3 RX, E RX F then* $\exists$ *RY, E RY G"*.

5.5.2 Sufficient Conditions

As stated in Section 5.2.3, if a SIC requires the existence of an occurrence of one relationship type to depend on the existence of an occurrence of the other relationship type, it would be incorporated when identifying necessary conditions for the existence of the occurrences of the first relationship type. At this step, we need not specially incorporate it as a sufficient condition for the existence of the occurrences of the other relationship type. However, other SICs may specify sufficient conditions for a relationship to exist involving the existence of an entity occurrence, time-triggering, or temporal sequence requirements. They are as follows.

The Existence of an Entity There may be SICs to require that if an entity occurrence exists, it must participate in a specified relationship type. In this case, the existence of a relationship occurrence might be thought of as a necessary condition for the existence of an entity occurrence. However, since "entity" is a more fundamental construct it is more natural to think that the existence of an entity occurrence is the sufficient condition for the existence of a relationship occurrence. There are some further complicating factors such as the following:

1. **The first complicating factor — implicit subtype.** It is possible that a relationship type must exist only for an "implicit entity subtype". In this case, the definition of an implicit entity subtype may be based on the values of its attributes[68].

[68] In this case, the only focus is the existence of an entity. We need not consider SICs related to an "implicit entity subtype" that is defined as occurrences participating in other relationship types. Those SICs have been covered when we consider necessary conditions for the existence of other relationship types.

2. **The second complicating factor — quantitative requirements.** We may require not only the existence of any occurrence of the specified relationship type, but also specify the number of such occurrences. These are two kinds of relative cardinalities[69]: the requirement of the minimum number of relationship occurrences in which an occurrence of the specified entity type must participate; or the requirement that for an occurrence of the specified entity type, all occurrences of the other entity type are required to connect with it via this relationship type. Combining this factor with the first factor, we would have cardinalities for an "implicit entity subtype"[70].

3. **The third complicating factor — further restrictions.** It is possible that there may be further restrictions on the existence of a relationship in addition to the cardinalities. A SIC may require that if an occurrence of one entity type exists, it must participate in some minimum number of occurrences of a specified relationship type, and there are further restrictions placed on the relationship attribute value(s), or on the occurrence(s) of the other participating entity type. SICs related to a weak entity type[71] and a **Critical_Relationship_Occurrence SIC**[72] are just special cases[73].

---

[69] Note that as described in the previous section, a relative maximum cardinality is a necessary condition for the existence of a relationship occurrence. The exact number of the cardinalities is the conjunction of the requirements of the minimum and maximum cardinalities.

[70] Again note that some subtype cardinalities, which are related to the entity participating in other relationship types, have been considered in the previous section.

[71] By the semantics of the weak entity, the relationship type R, via which the weak entity type is dependent upon a regular entity type, is total to the weak entity type and its key is fixed. The fixed retention restricts the key of relationship from updating. Note that the relative cardinalities of the weak entity in the relationship type R are not necessarily (1,1), it may be (c,d) where d is a number greater than c, and c greater than 1. For example, *a child as a dependent may have both father and mother as employees in a company.*

[72] A **Critical_Relationship_Occurrence SIC** requires the totality constraint to the specified entity type and the existence of exactly one critical relationship occurrence.

[73] There are many other examples such as **Stronger Totality Constraints** (refer to Appendix D).

4. **The fourth complicating factor — a group of relationship types.** We may not require the existence of an occurrence of a specific relationship type for any occurrence of the specified entity type. Instead, we may require the existence of one relationship occurrence among a group of relationship types. Combining this factor with the above factors, we might have a more complicated SIC.

Time-Triggering A single relationship, like an entity in isolation, may have time-triggered SICs to trigger its update.

Temporal Sequence Conditions It is possible to require that if an entity occurrence had some attribute value(s) in the past and no longer has it now, it must participate in some relationship(s). In the case of a group of relationship types, if some relationship(s) existed in the past, once it is deleted, other relationship(s) may be required to exist now.

5.6 SICs Implied by Implicit Relationships and Data Abstractions

In general, the E-R-SIC model requires us to express explicitly all relationship types in an E-R diagram. Even in cases involving an ID dependency[74], *is_a* or *component_of* relationships where the candidate key of one entity provides information about a related entity, the relationship type is required to be specified to make the semantics clearer. There are some exceptions (e.g., the cases that two entity types are exclusive; an entity type may be formed by set operations on other entity types; or *member_of* relationships except for in the enumerated set association) that there is a SIC between some entity

[74] An entity type (e.g., E) has "ID dependency" on other entity types if this entity type cannot be uniquely identified by its own attributes and has to be identified by its relationship types to the other entity types ([Chen, 1985]). The entity type E needs the key of some other entity type (e.g., F) as a part of its key.

types, but we need not express the relationship type explicitly because no suitable relationship type can be specified or the relationship is derivable. This section considers the SICs implied by these "implicit" relationship types. In addition, this section discusses the SICs implied by data abstractions since they are special relationships.

Specialization Abstraction The specification of a specialization abstraction, *"S is_a G"*, implies that the relative cardinality constraints of *is_a* must be *S (1,1)* and *G (0,1)* ([Goldstein and Storey, 1990]). Because the primary key of *G* (say *G.Gkey*) is a candidate key of *S*, there is a necessary condition, in addition to the existence of participating entity occurrences, for a relationship *is_a* to exist, i.e., *S.Gkey=G.Gkey.*

Entity Types in a Specialization Hierarchy In a specialization hierarchy, two entity types may be exclusive, or an entity type may be formed by set operations (intersection, union, or difference) on other entity types ([Biller and Neuhold, 1978])[75]. It is not natural to define a relationship to express the exclusion between two entity types. Neither is there an explicit relationship to express the set operations although *is_a* relationships among those entity types should be specified.

- **SICs implied by exclusion between entity types.** Suppose that two entity types *E, F* are exclusive and *Ekey* is their common candidate key. It implies a SIC written in the simplified format as: *if E.Ekey=Value then ¬(F.Ekey=Value).*

- **SICs implied by set operations on entity types.** Suppose *E* is formed by the operation on two entity types *F, G* and *Ekey* is the common candidate key among

[75]It is not meaningful to discuss exclusion or set operations if those entity types are not in the same hierarchy. In addition, if those entity types formed by set operations do not have special attributes or participate in special relationships, they may be redundant.

E, F and *G*.

- $E = F \cap G$. There are some SICs implied by the *is_a* relationships between the subtype (*E*) and the super-types *F, G*, respectively. In addition, it implies *"if (F.Ekey=Value)* $\wedge$ *(G.Ekey=Value) then E.Ekey=Value"*.
- $E = F \cup G$. There are some SICs implied by the *is_a* relationships between the the subtypes *F, G* and the super-type *E*, respectively. In addition, it implies *"if E.Ekey=Value then (F.Ekey=Value)* $\vee$ *(G.Ekey=Value)"*.
- $E = F - G$. There are some SICs implied by the *is_a* relationships between the subtype (*E*) and the super-type (*F*), and the exclusion between *E* and *G*. In addition, it implies *"if F.Ekey=Value then (E.Ekey=Value)* $\vee$ *(G.Ekey=Value)"*.

Inheritance Conflict Problem An inheritance conflict may occur when two or more super entity types of a single specialization type have some attribute(s) with the same name(s). In such a case, there should be a further higher level super-type of which these super-types are subtypes. However, the organization may not be interested in this higher level super-type. If these attributes are really semantically different, we just prefix their names with the super-type names and no SIC needs to be specified. However, if their semantics are the same, the subtype should inherit only one of these attributes and there must be an integrity constraint to maintain the equality of those attributes' values. In addition, the SICs associated with the super-types must not be inconsistent with each other, otherwise, the multiple inheritance will cause the subtype to be empty.

Aggregation Abstraction The specification of a composite entity aggregation, *C component_of A*, implies that the cardinalities of *component_of* should be either *C (1,1)*

component_of A (0,n) or *C (1,1) component_of A (n,m)*, where $n \geq 1$ and $m \geq n$[76]. Because a candidate key of C is formed by the primary key (say *A.Akey*) of A plus probably some other attributes, there is a necessary condition, in addition to the existence of participating entity occurrences, for a relationship *component_of* to exist, i.e., *C.Akey=A.Akey*.

Association Abstraction Association (grouping) is a powerful abstraction, but its implied SICs are complicated. In all cases except for the enumerated set association, the *member_of* is implicit since it can be derived. Their aggregate SICs should be specially dealt with[77]. Suppose that *agg_fcn* stands for an aggregate function, *Derived_Att* is an attribute of the set, *Att* is its corresponding attribute of the members.

1. **Natural Set Association:** Suppose that in the *M member_of Ms*, the *member_of* is implicit. We would have "*Ms.Derived_Att=agg_fcn(M.Att)*". In addition, an occurrence of *Ms* cannot be deleted if it, as a set, is not empty.

2. **Indexing Derived Set Association;**

   - **Only involving one indexing type:** Suppose that in Figure 5.1, for *M member_of DS*, the indexing entity type is *I* with the key *I.Ikey*, the indexing

[76] If the component *C (e.g., Automatic_window_controller)* is just relevant to the aggregate *A (e.g., Car)*, the aggregate *A* may have 0 or up to *n* components *C*. If the component *C (e.g., Engine)* is characteristic or identifying to the aggregate *A (e.g., Car)*, *component_of* is total to the aggregate *A*. (Reference to [dos Santos, et al., 1980] for the formal definitions of "relevant", "characteristic", and "identifying".) In either case, the *component_of* relationship should always be total to the component *C*. In some applications, some "components" can exist independently, e.g., *an engine may be sold as a separate part.* It would be better to define a subtype of the original "component" type to include those components that cannot exist independently so that the semantics become clearer. For example, we may have a type *Engine* and its subtype *Component_engine* that belongs to cars. Those components are different from other occurrences in terms of behavioral constraints and structural attributes.

[77] The cardinalities of *member_of* in the case of natural set association and indexed set association can be derived from the relative cardinalities of other relationship and absolute maximum cardinalities of entities. So, they need not be considered.

relationship is R, *member_of* is implicit. We would have:
"*DS.Derived_Att=agg_fcn({M.Att | M R I, DS.Ikey=I.Ikey})*".

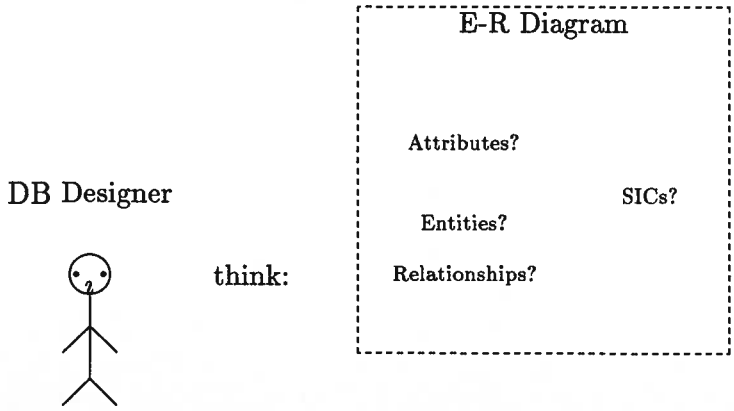
- **Only involving indexing attributes:** Suppose that in *M member_of DS*, *M.Index* is the indexing attribute, *member_of* is implicit. We would have: "*DS.Derived_Att=agg_fcn({M.Att | DS.Index=M.Index})*".

In both cases, an occurrence of *DS* cannot be deleted if it, as a set, is not empty. In addition, *DS.Ikey* (or *DS.Index*) cannot be updated.

3. **Enumerated Set Association:** In this case, *M member_of ES*, the *member_of* is explicit. That is, we have:
"*ES.Derived_Att=agg_fcn({M.Att | M M_member_of_ES ES})*".

5.7 Summary of the E-R-SIC Model

Figures 5.6 to 5.10 give a summary of the E-R-SIC model.



Construct Definition:

Definition 5.1: An **entity** is the database representation of a real-world object that can be distinctly identified.

Definition 5.2: A **relationship** is the database representation of an association among real-world objects.

Definition 5.3: An **attribute** is the database representation of a property of a real-world object or an association that is a function mapping an entity type or a relationship type to values.

Definition 5.4: A **SIC** is a logical, invariant restriction on the static state of a database (that is, a collection of attributes, entities, and relationships), or on the database state transition caused by an insertion, deletion, or update operation.

Observation:

- Entity, relationship, and SIC occurrences are classified into different entity, relationship, and SIC **types** according to some criteria.
- At a particular moment, certain groups of entity, relationship, or SIC occurrences can be considered as **sets** in the mathematical sense, and these sets may have some aggregate properties.
- Entity is the most fundamental construct among attribute, entity, and relationship.

Simplifying Assumptions:

— No ternary relationships or relationships of higher degree.
— No recursive relationships.

Data Abstractions: **inclusion** (classification, generalization), **aggregation**, and **association**.

Premise 5.1: A physical-symbol system has the necessary and sufficient properties to represent real-world meaning.

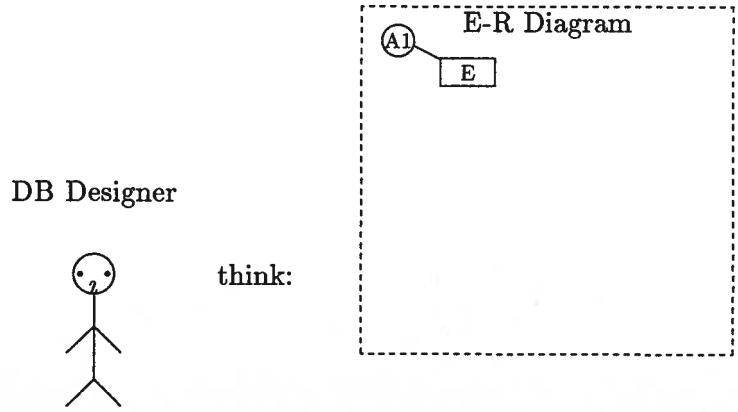
Interpretation: A SIC is an assertion — a sufficient or necessary condition — for an occurrence of an attribute, entity, or relationship type to exist, not exist, or change in a database.

Premise 5.2: The occurrences in each entity or relationship type are not homogenous; that is, they have different attribute values, and are related to different occurrences of other objects.

Implication: A database designer could not specify all entity and relationship types such that all occurrences of each type satisfied the same set of SICs.
That is, "implicit subtypes" with unique SICs always exist.

What are SICs that should be incorporated?

Figure 5.6: What is in the database?



Examine each attribute of each entity in isolation.
(That is, imagine a hypothetical situation:
examine each attribute, say *E.A1*, and suppose that it is the only "object"
of interest at this moment.)

Consider:

Are there necessary conditions that must hold
for *E.A1* to exist, not exist, or change in the database?

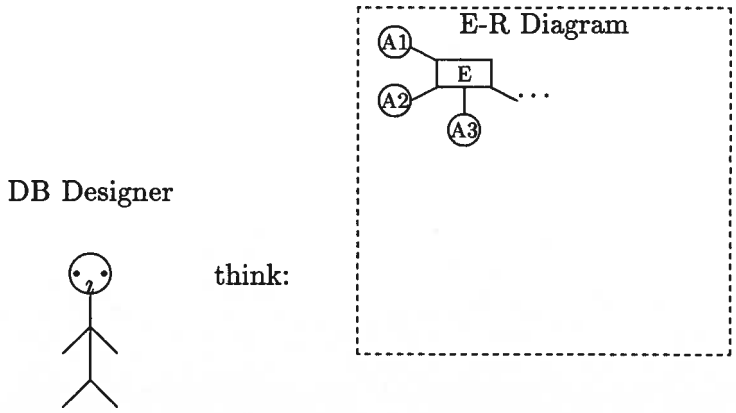
What are these necessary conditions:

- associated single entity type SICs?
  — existence: value, null, etc.?
  — change: changeable, new/old value pair?
- associated whole entity set SICs?
  — unique?
  — aggregate value?
- because we choose it as the primary key?
- because of time restriction?

Are there sufficient conditions, which if true, imply that
E.A1 must exist, not exist, or change in the database?

What are these sufficient conditions: none except for time-triggering?

Figure 5.7: Single Entity Attribute SICs



Examine each whole entity in isolation.
(That is, imagine a hypothetical situation:
examine each whole entity type, say E, and suppose that it is the only "object"
of interest at this moment.)

Consider:

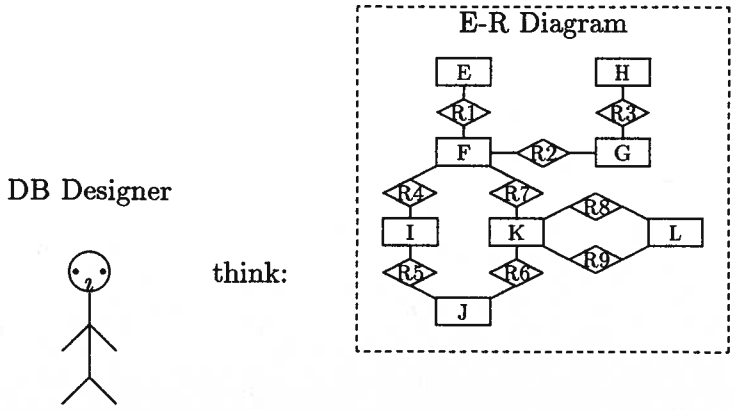
Are there necessary conditions that must hold
for an E occurrence to exist, not exist, or change in the database?

What are these necessary conditions:
- single entity type SICs?
  - — existence: a formula among attributes?
  - — change: deleted entity attributes?
- whole entity set SICs?
  - — the restriction on the maximal number of occurrences?
  - — concatenated values of some attributes are unique?
  - — a formula involving aggregate value(s)?
  - — aggregate values are interdependent?
- because of an implicit subtype?
- because of time-restriction?
- because of temporal sequence among attributes?

Are there sufficient conditions, which if true, imply that
an E occurrence must exist, not exist, or change in the database?

What are these sufficient conditions: None except for time-triggering?

Figure 5.8: Single Entity SICs



Examine each relationship type, say R, and consider its context in the whole E-R diagram.
What are the contexts that each relationship is in the E-R diagram: line, star, loop-2, and loop-n?

Consider:
What are necessary conditions that must hold
for an R occurrence to exist, not exist, or change in the database?

What are these necessary conditions:
- single relationship SICs?

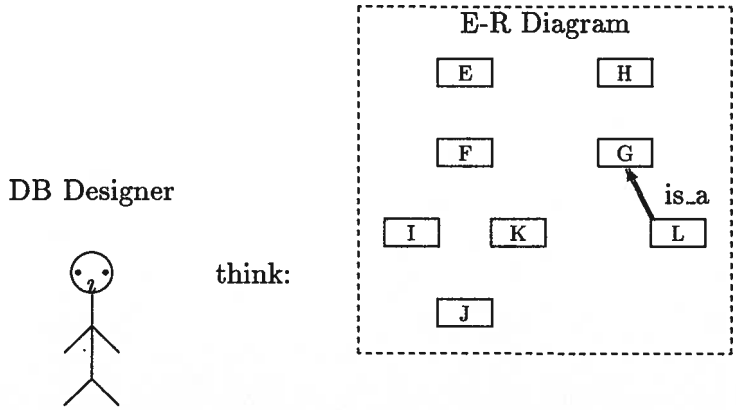
— similarly to a single entity, focus only on its attributes:
  * each of its single attribute has conditions in Figure 5.7?
  * the whole relationship has conditions in Figure 5.8?
    but could be more complicated since an implicit relationship type may be defined based on its participating entities or there may be a formula among those attributes of the relationship and its participating entity?

— focus on its role as an association between entities:
  * inherent SIC: its participant entity must exist?
  * restricted-connecting set:
    one-side, two-side of its participating entities, or intra-relationship condition?
    @ complicating factors:
      1. temporal conditions on relationships or the values of entity attributes?
      2. quantitative requirements on occurrences of the same or different relationship types?
      3. existence of multiple relationship occurrences of the same type?

- group relationship type SICs: conditions for occurrences of several relationship types to coexist?
- relationships involved in defining "implicit entity subtypes", so review Figures 5.7 and 5.8 again?

Are there sufficient conditions, which if true, imply that
an R occurrence must exist, not exist, or change in the database?
Some are covered as necessary conditions on other relationships, what are other sufficient conditions?
- the existence of an entity?
  @ complicating factors:
    1. because of implicit entity subtypes?
    2. quantitative requirements on the existence of the relationship occurrences?
    3. further restrictions on the existence of the relationship occurrences?
    4. the existence of one relationship occurrence among a group of relationship types?
- Time-triggering?
- Temporal sequence conditions?

Figure 5.9: Relationship SICs



Is there any SIC implied by some "implicit" relationships?

If there is a SIC between any two entity types,
in general we should express the involved relationship type explicitly in the E-R diagram.

In what situations would we have some "implicit" relationship types?

- these entity types are in the same specialization hierarchy?
— two entity types are exclusive?
— one entity type is the intersection of other entity types?
— one entity type is the union of other entity types?
— one entity type is the difference between two entity types?
— multiple inheritance conflict problem?

What are SICs implied by data abstractions?

- generalization: *S (1,1) is_a G (0,1)*,
  a necessary condition for *is_a* — *S.Gkey=G.Gkey*
- aggregation: *C (1,1) component_of A (0,n)* or *A (n,m)*, $n \geq 1$, $m \geq n$
  a necessary condition for *component_of* — *C.Akey=A.Akey*
- association and grouping: member_of may be implicitly represented except for
  the enumerated member_of;
  the involved relationships and entities are tightly related;
  SICs are complicated

Figure 5.10: SICs Implied by Implicit Relationships and Data Abstractions

Chapter 6

The Application of the E-R-SIC Model

This chapter begins with a hypothetical example to show how the E-R-SIC model can be applied. Potential pitfalls of using the E-R-SIC model are then discussed. Finally, we discuss the usefulness of the E-R-SIC model.

6.1 An Example of Using the E-R-SIC Model

Description Suppose that we have an example called *car_dealer_database_design.* The E-R diagram in shown in Figure 6.1. The database is intended to keep track of each car from the time it is ordered from the factory through its sale to a customer and subsequent service history as long as the car is serviced by that dealership. Information about a (potential) customer is also kept. For simplicity, the possibility that a customer might transfer his (her) car to other persons is not considered.

Attributes The attributes are not shown in the first level of the E-R diagram. They are as below[78].

- **Entity Types:**

[78]The property inheritance principle is assumed. Therefore, the inherited attributes, e.g., Salesperson.Salary, are not listed.

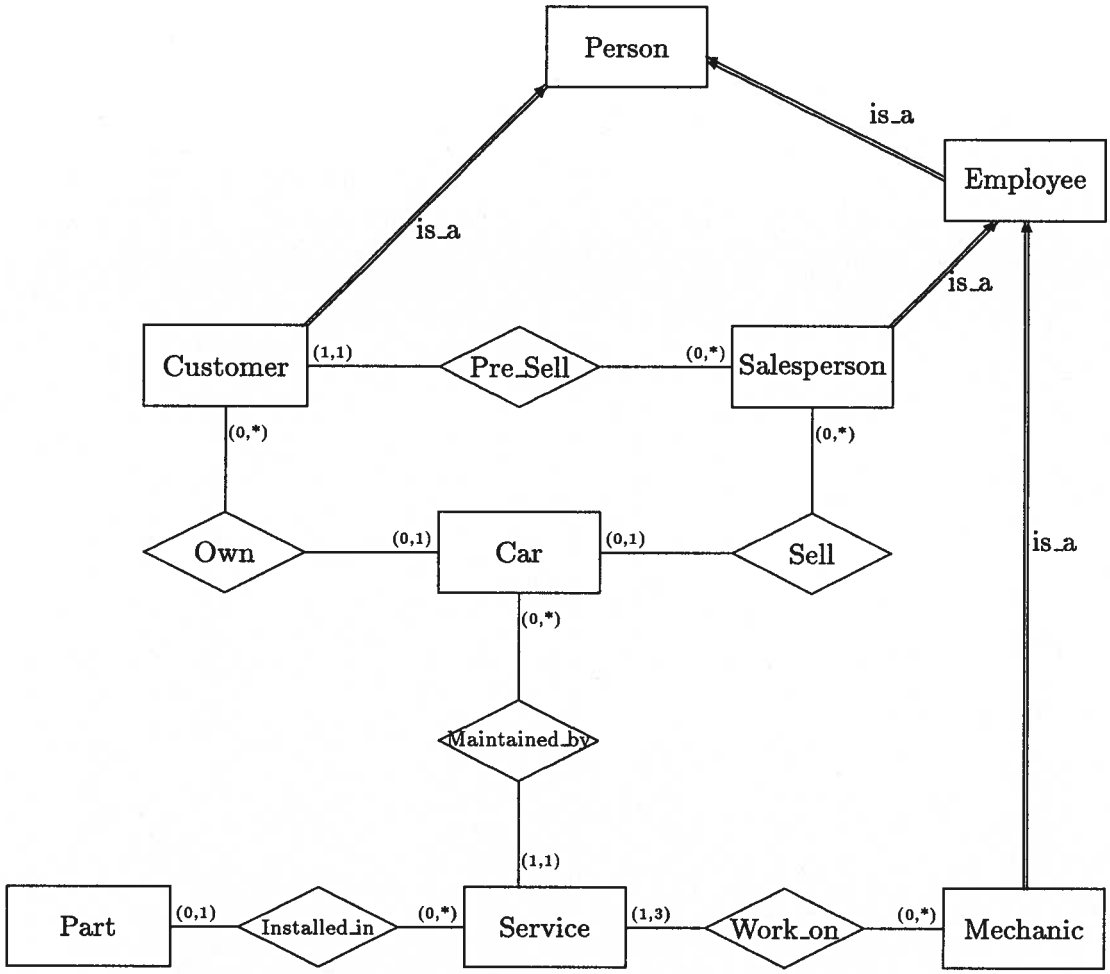


Figure 6.1: An Example: A Car Dealership Database

- **Person:** SIN (key, i.e., social insurance number), Name, Sex, Address, Phone.
- **Customer:** SIN (key).
- **Employee:** Emp_Id (key, i.e., employee identification number), Salary, Hire_Date.
- **Salesperson:** Emp_Id (key), Basic_Salary, Commission.
- **Mechanic:** Emp_Id (key).
- **Car:** Engine_No (key), Model, Year, Regular_Price, Cost.
- **Part:** Code (key), Name, Price, Cost, Qty_on_hand, Reorder_Point, Date_Last_Used.
- **Service:** Sequence_No (key), Date, Charge, Part_Fee, Labor_Fee, Discount_Amt.

- **Relationship Types:**
  - **Pre_Sell**: no attributes.
  - **Own:** no attributes.
  - **Sell:** Discount_Rate, Total_price, Sell_Date.
  - **Maintained_by:** no attributes.
  - **Installed_in:** Qty_Used (i.e., the number of the related part installed during a service).
  - **Work_on:** Hours (i.e., how much hours a mechanic spends on a service).

The example does not include all SIC types. In addition, only some of the relevant SICs are illustrated. The SICs are written in the simplified format according to the BNF in Appendix C.

Entity Attribute SICs Each attribute of each entity may have some SICs requiring it to be not-null, to take some specified value range, data type and format. The following two attributes are used for illustration.

- Person.SIN: there are SICs to assert that *a person's SIN must be known, is of type non-arithmetic (i.e., may not be used in conventional arithmetic operations) and has a format such as 123-456-789*, i.e.,

  is_null(Person.SIN)=no

  satisfy_datatype(Person.SIN, non_arithmetic)

  satisfy_format(Person.SIN, 999!999!999)

- Employee.Salary: there are SICs to assert that *an employee's monthly salary must be known, is of type arithmetic (i.e., all of the normal arithmetic operations may be performed on it in the usual way), and is in the range $1,500 and $10,000 inclusive*, i.e.,

  is_null(Employee.Salary)=no

  satisfy_datatype(Employee.Salary, arithmetic)

  satisfy_value(Employee.Salary, [1500 .. 10000])

Some attributes are not allowed to be updated. For example, *the social insurance number of a person cannot be updated*, i.e.[79],

if Person.SIN is_to_be_updated then false

Some attributes have SICs to restrict changes in value. For example, *a salesperson's basic salary cannot decrease*; and *the recently used date of each part is kept up-to-date*, i.e.,

new(Salesperson.Basic_Salary) $\geq$ old(Salesperson.Basic_Salary)

new(Part.Date_Last_Used) $\geq$ old(Part.Date_Last_Used)

[79]In this section, it is assumed that there is no automated database design aid. If there is such a system, the system, rather than the database designer, would write the SICs in the simplified format for later reformulating them in terms of SIC Representation model.

Some attribute SICs relate to the entire associated entity set, for example, *each person's SIN is unique*; and *the sum of the monthly salary of employees should be less than or equal to $1,000,000*, i.e.,

unique(Person.SIN)

sum(Employee.Salary) $\leq$ 1000000

Entity SICs There may be some SICs that require the values of several attributes of an entity to follow some formulas. For example, *the salary of a salesperson includes two parts — basic_salary and commission*; *the regular price of a car is set to have at least 25% gross profit*; and *for each part, its quantity on hand must be always greater than or equal to its reorder point*, i.e.,

Salesperson.Salary = Salesperson.Basic_Salary + Salesperson.Commission

Car.Cost $\leq$ Car.Regular_Price $\times$ 0.75

Part.Qty_on_hand $\geq$ Part.Reorder_Point

There are some SICs to restrict entity set properties. For example, *the dealer can have at most 30 mechanics*, i.e.,

count(Mechanic) $\leq$ 30

There are some SICs restricting attribute values as a function of time. For example, *for those mechanics who have worked for at least two years, their salaries should not decrease and the average should be at least $3,000*, i.e.,

if (Current_time $-$ Mechanic.Hire_Date) $\geq$ "2 years"

then new(Mechanic.Salary) $\geq$ old(Mechanic.Salary)

if (Current_time − Mechanic.Hire_Date) ≥ "2 years"
then avg(Mechanic.Salary) ≥ 3000

Relationship SICs Attributes of relationships may have similar attribute SICs. However, some restrictions on attributes of one relationship are more complicated. For example, *a salesperson cannot offer more than 80% of the gross profit rate of a car as a discount rate to a customer*; and *the total price of a sold car is equivalent to its regular price minus discount, then adding 13% tax rate*, i.e.,

if Salesperson Sell Car
then Sell.Discount_Rate ≤ (Car.Regular_Price − Car.Cost) ÷ Car.Regular_Price×0.80

if Salesperson Sell Car
then Sell.Total_Price = Car.Regular_Price × (1−Sell.Discount_Rate) × 1.13

There are some restrictions on constructing a relationship. Some restrictions are one-side conditions. For example, *the dealer only services those cars that are first owned by some customers that are in the database*, i.e.,

if Car Maintained_by Service
then (Customer Own Car) before

Other restrictions are two-side conditions. For example, examine the loop formed by the three relationships, *Pre_Sell, Sell* and *Own*, there are only three non-redundant SICs to assert their dependencies: (1) *if a salesperson sells a car, he (she) must have contacted a customer, who then buys the car — there must be a "customer" involved;*

any other "entity occurrences", which are not in the "customer type", cannot own that car; (2) *a customer can own those cars that have been pre-sold by a salesperson and then actually sold by the same salesperson*; (3) *if a car is sold by a salesperson and is owned by a customer then that salesperson must have contacted that customer — transfer of the car among customers is not recorded*, i.e.,

if Salesperson Sell Car
then Salesperson Pre_Sell Customer, Customer Own Car

if Customer Own Car
then Salesperson Pre_Sell Customer, Salesperson Sell Car

if Salesperson Sell Car
then if Customer Own Car
    then Salesperson Pre_Sell Customer

Some conditions are complicated. For example, *a part_fee including 6% tax is placed on the parts installed in those cars that have been bought more than 5 years*, i.e.,

if Part Installed_in Service,
    Car Maintained_by Service,
    Salesperson Sell Car,
    Service.Date − Sell.Date > "5 years"
then
    Service.Part_Fee = sum({MCharge | Part Installed_In Service,
                MCharge= Part.Price × Installed_In.Qty_Used × 1.06})

There can also be SICs expressing sufficient conditions for relationships. Those are the relative minimum cardinalities in Figure 6.1. For example, *any service should have at least one mechanic working on it*, i.e.,

$\forall$ Service, $\exists$ Work_on, Mechanic Work_on Service

SICs in a Specialization Hierarchy There are some SICs implied by implicit relationships. For example, *an employee cannot be both a mechanic and a salesperson*, i.e.,

if Mechanic.Emp_Id=Value
then ¬(Salesperson.Emp_Id=Value)

There are also some SICs implied by *is_a* relationships. For example, *"each mechanic is an employee"* implies that relative cardinalities — Mechanic (1,1) is_a Employee (0,1), and a necessary condition for an *is_a* occurrence to exist — *Mechanic.Emp_Id=Employee.Emp_Id.*

6.2 Potential Pitfalls of Using the E-R-SIC Model

There might be some potential pitfalls that need to be avoided when using the E-R-SIC model.

Pitfall 1: Inconsistent and Redundant SICs Inconsistent or redundant SICs might be specified. For instance, in the *car_dealer_database_design* example, if the database designer specified that *"if Salesperson Pre_Sell Customer then ¬(Salesperson Sell Car)"*, it

would be inconsistent with "*if Customer Own Car then Salesperson Pre_Sell Customer, Salesperson Sell Car*". If the database designer specified: "*Salesperson.Basic_Salary* $\geq$*2000*", "*Salesperson.Commission* $\geq$*500*", and "*Employee.Salary* $\leq$ *2400*", these SICs would be inconsistent because *Salesperson.Salary* would then have the lower bound \$2,500 that is greater than the upper bound of *Employee.Salary*. If the database designer specified "*if Customer Own Car then Salesperson Sell Car*", it is redundant since it is subsumed by "*if Customer Own Car then Salesperson Pre_Sell Customer, Salesperson Sell Car*". It becomes more difficult to detect such inconsistencies and redundancies as the number of SICs gets larger. There needs to be some automated tools to help detect them or prevent them in advance although this issue may not be completely solvable.

Pitfall 2: Imprecise or Incomplete SIC Representation The simplified representation format in the preceding section should only be used as a convenient way of initially incorporating SICs rather than as a formal way of eventually representing them. Although the simplified representation format might be more natural to a database designer, it is not precise. Recall that there are six components in the SIC Representation model. In most cases, the simplified representation only contains precondition and predicate components. For operation-dependent SICs, object and operation type components are also specified. Even in that case, other important specification information (the certainty factor and violation action) is still missing. In order to represent any SIC precisely, we need some algorithms to reformulate it (decompose it if necessary) in terms of the Representation model.

We do not take the naive approach of enumerating all conditions for all objects explicitly. For example, a general SIC represented in the simplified format, "*if E RX F then E RY G*", would be incorporated when we consider necessary conditions for the

existence of *RX*. We do not incorporate it again as a sufficient condition for the existence of *RY*. However, if we decompose the original general SIC, there would be two sub-SICs represented in the Representation model:

— one for *RX* on insertion, whose violation action would be "reject" or "propagate(insert(RY))"

— the other for *RY* on deletion, whose violation action would be "reject" or "propagate(delete(RX))"

Note that the first sub-SIC in terms of the SIC Representation model is a necessary condition for the insertion of *RX*, and is also a sufficient condition for the insertion of *RY* if the violation action is "propagate(insert(RY))". The second sub-SIC is a necessary condition for the deletion of *RY*, and is also a sufficient condition for the deletion of *RX* if the violation action is "propagate(delete(RX))". Without decomposing a general SIC into several sub-SICs represented in terms of the SIC Representation model, the SIC specifications would not be complete because the violation action may depend on the object and operation causing violation. Therefore, we can conclude that the SIC Representation model provides a complete and precise representation of SICs in the E-R-SIC model.

Pitfall 3: E-R Structure Orientation The E-R-SIC model helps a database designer incorporate SICs based on E-R structural descriptions. However, our logical data model is the relational model. There should be some procedure for losslessly transforming SICs in the E-R-SIC model into corresponding ones in the relational model[80].

[80]The "losslessly" means that, for each SIC incorporated in the E-R-SIC model placing restrictions on occurrences of entities, relationships, or attributes, there is (are) SIC(s) in the relational model placing corresponding restrictions on relation tuples or attributes.

Pitfall 4: Inefficient Modelling In the absence of an automated database design aid, a database designer must specify all explicit and inherent SICs in terms of the simplified formats as described in Appendix C. For example, in addition to specifying the entity types, *Employee* and *Mechanic*, and the relationship type *Mechanic_is_a_Employee*, he (she) needs also to specify the relative cardinalities of *is_a*, a necessary condition for its existence — *Mechanic.Emp_Id=Employee.Emp_Id*, and even the fundamental incidence constraints — the existence of participating entity occurrences. In general, there might be many possible SICs given an E-R diagram. Since one SIC may involve more than one object, the database designer needs to record what SICs have already been identified so that they would not be incorporated again. There would be a heavy burden for the database designer if he (she) needs to figure out all the possible SICs, verify, reformulate, decompose, and transform them by himself (herself). It is desirable to have some automated database design system to help the database designer model and represent SICs.

Proposals for avoiding these pitfalls will be introduced in the next chapter (pitfall 1 is not completely avoided).

6.3 Data Integrity Semantic Completeness

In the E-R-SIC model, the SIC is the construct used to provide logical restrictions on the structural schema, that is, on the attributes, entities and relationships. Completely incorporating SICs depends on the correct specification of the structural schema. The E-R-SIC model requires all relationships to be explicitly represented except for some special cases, such as exclusion between two entity types. The explicit representation of relationships makes the semantics clear.

Based on the assumed correct E-R structural schema, we could systematically model SICs by considering a wide range of possible restrictions or requirements — positive or negative, simple assertions or complicated formulas, qualitative or quantitative, time-restricted, time-triggering, or temporal sequence requirements, implicitly/explicitly type-related or set-related. Most SICs we consider are necessary conditions for the existence of the occurrences of one or more attribute, entity, or relationship types. They are, by nature, static SICs restricting the possible states of the database. A few are inherently dynamic SICs, e.g., new_old transitional constraints; and some others are considered as sufficient conditions. However, after decomposing them into sub-SICs in terms of the SIC Representation model, all become dynamic and operation-dependent SICs specifying conditions for the deletion, insertion, or update of the related objects. Since these three operations are the only ways the database can change, those necessary conditions for these operations completely cover the necessary conditions for existence, nonexistence and change of an object in a database.

In addition, sufficient conditions (if any) are also incorporated because of the violation action component of the SIC Representation model.

Whenever we find that the structural schema cannot model the data integrity semantics as SICs, we would need to go back and modify the E-R structure. Therefore, the E-R-SIC model not only allows modelling the SICs completely but also forces the structural schema to reflect the data integrity semantics more faithfully. The structure and SIC parts of the resulting schema would be closely related. This would form a more suitable and fundamental base for higher level transaction or application program modelling. Therefore, we can conclude the following:

With the support of the SIC Representation model, the E-R-SIC model would

completely model the data integrity semantics.

There is one limitation. Although some entities or relationships may have time-valued attributes or temporal sequence requirements, time-stamped data integrity semantics (e.g., dealing with historical data at some specific time) in general may not be completely modelled[81].

A related discussion concerns SICs in the relational model. Suppose that the SICs in the E-R model have been losslessly transformed into corresponding ones in the relational model. Are there any other new SICs, e.g., data dependencies, that must be considered in the relational model?

Data Dependency in the Relational Model The relational data model has well-formalized theories. *Data dependency* is one of its important concepts. Ullman [1982] defines a data dependency as "a constraint on the possible relations that can be the current value for a relation scheme". However, one should note that the usefulness of data normalization theory is for designing a database directly from the relation concept, not for capturing semantics. The partial and transitive functional dependencies imply the embedding of independent relationships ([Makowsky, et al., 1986]). Similarly, the existence of non-trivial multi-valued dependencies occurs when a relation represents more than one 1:N relationship ([Kent, 1983]). These problems can be avoided if entities and relationships are properly designed and carefully transformed into relations ([Storey, 1988]). Some researchers try to apply other dependencies, e.g., inclusion dependencies,

[81] In the literature, [Palmer, 1982] discusses time-dependent relationships and [Tauzovich, 1991] introduces lifetime cardinalities. It is not clear how the restrictions of lifetime cardinalities are applied over the lifetime of the entities and relationships.

exclusion dependencies and co-exclusion dependencies[82], to capture semantics. Exclusion dependencies have been covered in the E-R-SIC model. Inclusion dependencies come about because some relationships have not been explicitly represented[83]; and co-exclusion dependencies occur because the SICs are not precisely represented[84]. Therefore, we need not add other SICs after transforming those SICs identified by the E-R-SIC model to corresponding ones that reference the schema in the relational model.

[82]Let E, F be two relations (possibly the same), and A_i, B_j, be attributes of E and F, respectively. $E[A_1, \ldots, A_m] \subseteq F[B_1, \ldots, B_n]$ is called an *inclusion dependency* ([Casanova and Vidal, 1983]). If $E[A_1, \ldots, A_m]$ is exclusive to $F[B_1, \ldots, B_n]$, it is called an *exclusion dependency* ([Casanova and Vidal, 1983]). The negation of an exclusion dependency is called a *co-exclusion dependency*, ([Arisawa and Miura, 1986]).

[83]Take an example of [Minnila and Räihä, 1986]: *Registered_Cars[Model]* $\subseteq$ *Car_Types[Model]*. There should be some relationship, e.g., *Has_model*, between *Registered_Cars* and *Car_Types*.

[84]Arisawa and Miura [1986] state that what co-exclusion dependencies mean is to constrain database when updating and sharing entities. However, there should a higher level of entity that is the super-type of those entity types. If the SICs implied the *is_a* relationships between the sub-types and super-type are precisely represented, the co-exclusion dependency constraints would be incorporated.

Chapter 7

A Proposed Database Design Aid for Eliciting SICs

In Figure 1.1 of Chapter 1 a proposed database design subsystem for eliciting SICs is sketched. Both the SIC Representation model and E-R-SIC model are application-domain independent. They are suitable for implementation as part of an automated database design system. An automated database design system is needed because of the potential heavy work load of a database designer when incorporating and representing SICs. This chapter describes conceptually how this subsystem for eliciting SICs should work. In the following, the term "database designer" will be used for the user of the automated database design subsystem who may be a professional database designer or an end-user who plays the role of the database designer to design his (her) system. Section 7.1 gives a brief overview of the elicitation subsystem. The three major functions of the subsystem — verifying elicited SICs for consistency and non-redundancy, reformulating and decomposing general SICs into sub-SICs in terms of the SIC Representation model, and transforming them into corresponding ones in the relational model are described in detail in Sections 7.2, 7.3 and 7.4, respectively.

7.1 An Overview of the SIC Elicitation Subsystem

Interfaces From the view of the *SIC elicitation subsystem*, there are two interfaces. The first is an interface to elicit explicit constraints from the database designer in a dialogue or by menus, etc. The second is an interface between the *SIC elicitation subsystem* and the *structure subsystem* (e.g., the View Creation System in [Storey, 1988] and [Storey and Goldstein, 1988]) for constructing the structural schema. From that interface, the *SIC elicitation subsystem* fetches the structural descriptions in terms of the E-R-SIC model.

Elicitation Based on these structure descriptions, the *SIC elicitation subsystem* will query the database designer to obtain the attribute SICs for each attribute of each entity. Next, whole entity SICs would be obtained for each entity. The system then traverses the E-R diagram to detect possible relationship SICs[85] and data abstraction SICs. In a dialogue, the database designer may need to add some further restrictions on a SIC by using expressions such as those in the simplified format (Appendix C). In general, though, the database designer need not worry about the assertion representation syntax. In addition, the system, rather than the database designer, would specify those inherent SICs implied by special relationships (e.g., the *is_a* data abstraction).

Elicitation Knowledge It is possible that the elicitation procedure may be lengthy because the goal is to include the complete data integrity semantics and the system has neither common sense nor application domain knowledge. However, it is expected that

[85]The original structural specifications may include some traditional relative cardinalities of relationships. However, the exact numbers except for 0, 1 may not be known since they are irrelevant for constructing normalized structures.

the system should query the database designer based on the following knowledge:

- the **E-R-SIC model** to capture the possible SICs from different objects in the (line, star, loop-2, and loop-n) contexts of an E-R diagram;
- **consistency and nonredundancy rules** for different SIC types;
- **heuristics** from naming conventions and data types;
- **specification information** that the database designer has provided so far.

With consistency and nonredundancy rules for different SIC types, the subsystem would not request, or could refuse immediately, some impossible SICs. This kind of knowledge is introduced in Section 7.2.1 to show how it can alleviate the need for some detailed verification.

Since there may be a large number of possible SICs, heuristics may help the subsystem to be even more efficient in asking questions. The object data type — date, arithmetic or non-arithmetic may suggest its volatility. For example, often a date is unchangeable. In addition, naming conventions may hint at some SICs. For example, attributes with the same names in the entity types linked via some relationship types may suggest the need for SICs involving some formula between them. Appendix E contains more examples.

Verification Although full verification for consistency and non-redundancy is hard to achieve, the elicited SICs would be verified to some extent, especially for cardinalities. The cardinality information is important because a totality constraint might make many possible SICs redundant or inconsistent. Section 7.2 discusses the related issues on verification in detail.

Reformulation and Decomposition Based on the information obtained from the database designer, the system would represent general SICs in terms of the simplified format described in Appendix C. Then, the system should automatically rewrite each of the above SICs, and decompose it into sub-SICs if necessary, in terms of the SIC Representation model. All six components plus the SIC name in the model would be specified. The reformulation and decomposition algorithms are introduced in Section 7.3. The default certainty factor is "certain", but the database designer can change it to be "uncertain" or provide specific certainty factors.

Without application domain knowledge, it is impossible for an automated database design system to provide automatically complex and application-dependent violation actions. For example, if we allow arbitrary violation actions, even for a SIC on update of *E.A2* with a simple formula predicate *"E.A1 = E.A2 + E.A3"*, the violation action may be either a propagation to update *E.A1*, or update *E.A3*, or update both *E.A1* and *E.A3* (this may imply a number of choices with different ratios of *E.A1* and *E.A3*). The number of possible combinations of violation actions would grow rapidly as the number of involved objects increases. The "propagate" action is helpful and powerful, but is also costly if the set of SICs is not optimized[86]. In this research, it is envisioned that the system would only provide some restricted predefined violation action choices ("reject", "propagate", "conditionally_reject", "conditionally_propagate", or "warning"). Most of the violation actions would be "reject", "conditionally_reject", or "warning" if the system could not decide how to propagate to fix the violation. If the system decides that a "propagate" action is feasible, it would query the database designer to determine

[86] An example here consists of two SICs: *"SIC-1: (E RX F) ∨ (E RY G)"*; *"SIC-2: if E RX F then E RY G"* attached with "propagate" actions in their all sub-SICs. Suppose that the original database state is "*RY exists, but RX does not*". A *"delete(RY)"* operation would cause the following operations to be performed: *"delete(RY)"*, *"propagate(insert(RX))"* owing to *SIC-1*, *"propagate(insert(RY))"* owing to *SIC-2*. A *RY* is forced to be inserted back although now it may connect *E* with a different *F*.

whether to "propagate" or "reject".

Generic SICs As described on page 78, we can represent some common SIC types for the generic object (*Entity**, *Relationship**, *Entity*.Attribute**, and *Relationship*.Attribute**) to reduce the number of explicit SICs represented using the Representation model. By doing so, we can also alleviate the need for the invocation of the reformulation and decomposition algorithms during the database design consultation session. The subsystem should have these pre-defined representations of generic SICs. If the database designer specifies that occurrences of an object type must satisfy one of these common SIC types (e.g., two entity types are mutually exclusive), the related constraint information would only be kept in a logical predicate, called an *input predicate* (refer to Appendix B.1, e.g., *ex_ents(ExEntSet)* storing the information that the entity types in *ExEntSet* are exclusive). Such specific constraint information need not be directly reformulated or decomposed in terms of the SIC Representation model. It is expected that the specified object type could inherit the related generic SICs. Section 7.3.1 describes these generic SICs in more detail.

Transformation It is assumed that if a relationship type has (1,1) cardinalities relative to an entity type, it would be represented by a foreign key rather than a separate relation in the relational model. When the *structure subsystem* transforms the E-R specifications into relations, the *SIC elicitation subsystem* should also transform the SICs in the E-R-SIC model into corresponding ones in the relational model. The input predicates need not be transformed. There should also be pre-defined generic SICs in the relational model corresponding to those in the E-R-SIC model. The principles of representing relationships in the relational model and the SIC transformation algorithms are introduced

in Section 7.4.

Final Outputs The final results of the consultation session would be a listing of incorporated SIC specifications in a relational database schema. These specifications include: (1) specific SICs, which directly apply to specific object types and are represented in terms of the Representation model; (2) input predicates describing specific object types to which some common SIC types apply; (3) generic SICs, which serve as the "templates" for common SIC representations and are expected to be inherited by the specific object types described in (2).

7.2 SIC Verification

Difficulty in Verification As discussed in Section 2.3, full verification of SICs for consistency and non-redundancy would need application-domain and common-sense knowledge. Even those aspects of verification that do not require application-domain knowledge still pose difficult research problems because of the potential complexities of SICs. This research discusses the issues, but does not offer general solution algorithms.

Because the interactions among SICs might, in general, be complicated[87], we would need logic theorem proving techniques (e.g., the resolution refutation method [Nilsson, 1980]). However, standard logic must be enhanced to handle the following problems.

- **Attribute Value Problem.** In standard logic programming theory, objects are

[87]For example, suppose that *C1, C2, C3* are the preconditions of three SICs, *SIC-1, SIC-2, SIC-3*, respectively; and *P1, P2, P3* are their predicates, respectively. *SIC-1* and *SIC-3* are asserted for the same object. If *C1* overlaps with *P2* and *C3* implies *C2*, *P1* should be the same as *P3*. Otherwise, they are inconsistent — because *C3* implies *P2* and *P2* overlaps with *C1*, if some occurrences satisfy *C3*, they may also satisfy *C1*; so *C3* transitively implies *P1*.

represented as finite terms, and the set of terms is countable (within the context of the Herbrand Universe) ([Lassez, 1987]). Suppose that the value domains of all attributes are **discrete and finite**. Then, consistency among SICs can be checked by applying the standard logic proving technique although, for efficiency reasons, it may be accompanied by the standard CSP (constraint satisfaction problem) algorithms ([Mackworth, 1977; 1987]; [Mackworth and Freuder, 1985])[88]. However, suppose that the values of attributes are allowed to be **continuous and infinite**[89]. Standard logic cannot deal with the real number domain. It must be extended to have other techniques to handle real number values and formulas. That is the motivation for developing constraint logic programming (CLP) ([Jaffar and Lassez, 1987]; [Cohen, 1990])[90]. We may apply linear programming (LP) algorithms (e.g., the simplex method [Hillier and Lieberman, 1986]) to check the consistency of a set of linear formulas (i.e., whether a feasible solution exists) and remove the inconsistent value ranges (by finding the values of optimization functions — maximization and minimization for each involved attributes)[91]; apply integer linear programming algorithms if the values are restricted to be integer numbers[92]; and

[88]There are three sets of algorithms — node, arc, and path algorithms. In this case, we would apply arc algorithms to check the consistency between value constraints and a formula constraint; apply path algorithms to check the consistency among value constraints and several formula constraints. They can be checked in polynomial time ([Mackworth and Freuder, 1985]) although the path algorithms are complicated.

[89]They may be numerical real numbers or integers; or may have date data type if date operations are suitably defined.

[90]For example, one CLP representative, CLP(R) provides both expressive and computational powers to solve linear equations and linear inequalities (by two-phase simplex algorithm) incrementally ([Jaffar and Michaylov, 1987]). However, it is still a rudimentary experimental product and can only run under UNIX-based operating systems. Other CLP languages are Prolog III ([Colmerauer, 1990]), Trilogy, and CHIP ([Hentenryck, 1989]), etc.

[91]A simple algorithm used by ALICE (A Language for Intelligent Combinatorial Exploration) ([Lauriere, 1978]) can also be applied to test the consistency between a single linear formula and the value range constraints, and further to remove the inconsistent value ranges.

[92]It is not usual to have complex numbers in a database. Suppose we do have a complex number application, Gröbner method would test whether a system of multivariate polynomial equations has a solution over the complex numbers ([Cohen, 1990]).

apply non-linear programming algorithms to handle a set of non-linear formulas[93]. However, all existing methods have some limitations. For example, if any of the connectives between linear formulas is a disjunction ("∨"), e.g., *"P1 ∨ P2"*, the simplex method is no longer applicable since *"P1 ∨ P2"* may represent a non-convex polyhedron ([Cohen, 1990])[94]. The general integer linear programming problem is NP-complete ([Papadimitriou and Steiglitz, 1982])[95]. Furthermore, it is even difficult to achieve nonredundancy. We may apply these algorithms to remove the inconsistent values from value constraints. However, given a set of arbitrary formulas, no existing algorithms can tell us whether a formula is redundant.

- **Aggregate Function Problem.** There may be some attribute SICs related to entire entity or relationship set properties. We cannot verify these SIC specifications without actual data. Even if they are not in rule format, at the best we can only have some simple tests (Appendix F.1), such as: the specified maximum value of an attribute value must be greater than or equal to its minimum value. In addition, as stated, the verification for cardinality constraints is important for other relationship SICs. Standard logic proving should be supplemented with the algorithms described in Appendix F.2 to check consistency and nonredundancy for cardinalities.

- **Incremental Verification Problem.** Verification would be conducted for the SICs in the sequence of the incorporation. Since it is desirable to identify any inconsistencies each time a new SIC is added, the *SIC elicitation subsystem* would

[93] If there are non-linear formulas and the functions are monotonically increasing or decreasing for each involved attribute, the ALICE algorithm can be applied to check the consistency between a single non-linear formula and value constraints.

[94] In addition, one should note that the simplex algorithm would take an exponential number of steps in the worst case although in general it can be regarded as very efficient ([Papadimitriou and Steiglitz, 1982]).

[95] It is known that the theory of predicates with $=, \leq, \geq, <, >, \neq, +, \times$ over real numbers is *satisfaction-complete*, i.e., every constraint is either provably satisfiable or provably unsatisfiable. However, the theory of predicates with $=, +, \times$ over integer numbers is not satisfaction-complete ([Cohen, 1990]).

not verify the whole set of SICs only once, instead there would be an incremental constraint satisfaction problem ([Hentenryck, 1990]), in which constraints are incrementally added and dropped.

- **Feedback from the Database Designer.** It is impossible for the *SIC elicitation subsystem* to achieve complete verification without feedback from the database designer. The following are two examples.
  - If the preconditions of two SICs overlap, the database designer is required to modify these two SICs.
  - If the preconditions of two SICs are not related in the syntax (i.e., they are expressions on different objects, for example, one is on *Employee.Age*, another is on *Employee.Education*), but their predicates are inconsistent, the system should query the database designer as to whether it is possible that an object would satisfy these preconditions at the same time. If it is, these two SICs are inconsistent.

Optimization Problem Even if two constraints are neither inconsistent nor redundant, they might not be "optimized" — i.e., there might be a single constraint that can replace the set of these two constraints and be enforced more efficiently. The following are two examples.

- Given two value range constraints that are neither inconsistent nor redundant, they may overlap. That is, there may be a single refined range to replace those two ranges.
- A set of two constraints, "*(E RX F)* $\vee$ *(E RY G)*", and *"if E RX F then E RY G"*, is neither inconsistent nor redundant. However, after optimization, the above set

would be replaced by a single constraint, *"∀ E, ∃ RY, E RY G"* (i.e., *RY total to E*).

To get a single refined range constraint may be easy. However, in general, optimization is more difficult than consistency and non-redundancy checking even for a small and simple set of constraints. For its related objects, each constraint allows a set of database states to exist. When several constraints relate to the same object, if they are consistent, the intersection of all those sets of states is not empty. One approach to optimization is to find this intersection set and describe it by some expressions. However, the number of states could be large and it would be very difficult to write expressions for those states.

Issue of Unexpected Implicit Constraints Even if a set of SICs is not inconsistent or redundant, there may still exist some unexpected implicit constraints. For example, suppose that *Supervise* is a relationship type between entity types *Manager* and *Project*, the database designer has specified the absolute maximum cardinality of *Project* — there can be no more than 8 projects; and some relative maximum and minimum cardinalities in the relationship *Supervise* — there can be no more than one manager per project and any manager must be assigned to a project. This set of three constraints implies an implicit constraint — the organization can hire at most 8 managers although the database designer does not specify it explicitly or may think that there is no upper bound on the absolute maximum cardinality of the entity type *Manager*. The *elicitation subsystem* needs to recognize that some SIC types may be missing, and then check the current set of constraints to see whether such types of SICs can be derived from it. The database designer would then be asked to judge as to whether he (she) has made some mistakes.

Verification related to Certainty Factors The certainty factor component in the SIC Representation model is a source of possible inconsistency between SICs. Supposing that both *SIC-1* and *SIC-2* are for the same object on the same operation with the same precondition, if the predicate of *SIC-2* implies that of *SIC-1* (i.e., *SIC-2* is more restrictive than *SIC-1*), the *SIC-2* should be less certain than *SIC-1*. Otherwise, they are inconsistent. In addition, the certainty factor and violation action are closely related. An uncertain SIC can only have a predefined violation action of "warning", "conditionally_reject", or "conditionally_propagate". A certain SIC can only have a predefined violation action of "reject" or "propagate".

Verification related to Violation Actions Furtado et al. [1988] describe a situation where the violation actions of two SICs may be inconsistent. However, if the SICs are precisely defined and the transaction concept is applied when enforcing them, the problem described by Furtado et al. could not occur[96]. The major consistency problem caused by the violation action component in the SIC Representation model is that the violation actions may cause an endless enforcement cycle — a "flip-flopping" behaviour between a set of SICs described by Ceri and Widom [1990] if arbitrary actions are allowed. Ceri and Widom suggest using a *triggering graph*[97] to detect potential cycles. However, they leave the responsibility of determining whether infinite triggering would actually be possible to the database designer. It is difficult, in general, to take an arbitrary violation action

[96]The problem raised by Furtado et al. [1988] is as follows. Suppose that there is a relationship type R and an entity type E, R is total to E. If the violation action of the incidence constraint, *SIC-1*, for E on deletion is "propagate(delete(R))" and the violation action of the totality constraint, *SIC-2*, for R on deletion is "reject", Furtado et al. claim that they are incompatible. However, note that if the SICs are precisely defined, when the *SIC-1* has been violated and the deletion of the R is taken, the *SIC-2* becomes irrelevant because the E occurrence has become non-existent.

[97]In their triggering graph, the nodes of the graph correspond to the SICs. There is a directed edge from node *SIC-1* to node *SIC-2*, if and only if the execution of *SIC-1*'s violation action is likely to violate *SIC-2* (the likelihood is only in terms of objects and operation types).

and decide whether it would violate other SICs. In this research, only some restricted and predefined violation actions would be suggested by the *SIC elicitation subsystem.* Basically, if the violation action of a SIC is "propagate", when the SIC is violated, it would allow the intended operation to be performed, but would also take a simple compensatory action to bring the database to another consistent state. With such a restriction, if those SICs are consistent (in terms of the components other than the violation action), when an operation violates a SIC, the database would return to a consistent state although some operations may be undone.

Verification and Decomposition If a general SIC is not in rule format, it can be verified for consistency and nonredundancy before decomposing it. However, a general SIC in rule format must first be decomposed into several sub-SICs before verifying it. These sub-SICs indicate the consequences of the general SIC that must be taken into consideration when verifying a set of SICs. For example, if there is a general SIC, *"if C1 then P1"*, where *C1* and *P1* are some assertions, the *SIC elicitation subsystem* would not know that it is inconsistent with *"if* $\neg$ *P1 then C1"* without first decomposing it. The decomposition algorithms described in the next section can be assured to represent correctly the sub-SICs of a given single SIC. However, the verification for consistency and nonredundancy among the representations for all SICs is still needed.

7.2.1 Consistency and Nonredundancy Rules for SIC Types

Since conceptually by using the E-R-SIC model, the SICs can be classified into a number of types, some general problems of consistency and nonredundancy among those types of SICs can be explored in advance. The results would be some rules about consistency and nonredundancy for SIC types, which can be stored in the knowledge base of the

elicitation subsystem in order to expedite the elicitation procedure.

Take a simple example. In Figure 6.1 of the *car_dealer_database_design* example, the two relationship types, *Pre_Sell* and *Own*, may be involved in a number of types of SICs. For example, they may be exclusive (i.e., *"if Customer Own Car then ¬(Salesperson Pre_Sell Customer)"*), or one may depend on the other (e.g., *"if Customer Own Car then Salesperson Pre_Sell Customer"*). Suppose that the database designer has specified that the minimum cardinality of *Pre_Sell* relative to *Customer* is 1. With the built-in consistency and nonredundancy rules in Appendix G, the *SIC elicitation subsystem* would automatically know that the first SIC type cannot occur and that the second SIC type is redundant. It might be confusing to the database designer if the *SIC elicitation subsystem* asks him (her) to confirm these SICs. It would be inefficient if the database designer inadvertently specified the above redundant or inconsistent SICs and the *elicitation subsystem* invoked the whole process to verify them during a consultation session.

Note that this knowledge cannot totally replace the actual verification during the consultation, but it would reduce the number of cases that need to be verified. Kung [1984, 1985] applies a sophisticated tableaux approach to verify a set of two SICs: *"every employee earns more than $20,000"*, *and "every manager is an employee"*. Since the first is a simple value constraint for a non-key attribute and the second implies some special relative cardinalities and the equality condition on key values of two sides of entities for the existence of an *is_a* occurrence, a human database designer would know that they cannot be inconsistent. If the *SIC elicitation subsystem* had similar rules, it could also skip the detailed consistency checking process.

7.3 SIC Reformulation and Decomposition

As discussed in Chapter 6, the E-R-SIC model uses the Representation model to specify SICs. SICs should be represented in terms of the Representation model. If a SIC is only relevant to an object on an operation, we only need to reformulate it in the Representation model without changing its certainty. In most cases, because a SIC is relevant to several objects on various operations, the decomposition would get several sub-SICs written in terms of the Representation model.

The *SIC elicitation subsystem* would invoke the reformulation and decomposition algorithms, which are described in Appendix H, to reformulate and decompose SICs obtained from the database designer. By using the E-R-SIC model, the *subsystem* should recognize all inherent SICs that are implied by the specifications (e.g., an *is_a* relationship) provided by the database designer. Before invoking the reformulation and decomposition algorithms, the *subsystem* should first represent these explicit and inherent SICs in terms of the simplified formats in Appendix C, but without using the nested *if* ... *then* rules.

Algorithms previously proposed in the literature can deal only with some restricted types of SICs and only consider the object and operation type components. The algorithms in Appendix H deal with all SICs that are recognized by the E-R-SIC model. All components in the SIC Representation model are considered.

Correctness of the Reformulation and Decomposition Algorithms It is necessary to assure the correctness of the proposed reformulation and decomposition algorithms for representing a given general SIC. The following briefly examines the algorithms.

1. **Same Certainty Factor.** The certainty of sub-SICs cannot be lower than the general SIC; otherwise, the enforcement of these sub-SICs cannot achieve the certainty of the general SIC. Therefore, if the original general SIC is 100% certain, its sub-SICs should also have 100% certainty. Suppose that the original general SIC is uncertain. Since its sub-SICs may have different violation actions, some of them may have higher certainty. Without further information, the *elicitation subsystem* can assume that they have at least the same certainty as the general SIC. (Later, the database designer can change the certainty of a sub-SIC to be higher if necessary.)

2. **Relevant Object and Operation Types.** The search for the relevant object and operation types is based on the following ideas.

   - An operation-dependent SIC is only relevant to one object on one operation. For example, *"if E is_to_be_deleted then ..."* is only relevant to E on deletion.
   - It is desirable to remove obvious redundancies when checking objects mentioned in a SIC. By the nature of relationship, if a relationship occurrence exists, its participating entity occurrences must exist too (the incidence constraint). If a general SIC is relevant to the insertion of an occurrence of a relationship type and we have a sub-SIC for the relationship, we need not have other sub-SICs to restrict the insertion of the occurrences of its participating entity types. It is possible that some occurrences of these entity types may not participate in any occurrence of this relationship type. The checking of the general SIC for any occurrence of these entity types can be delayed until it really participates in an occurrence of this relationship type. For example, according to the algorithms, *"if E RX F then E RY G"* would be relevant to RX on insertion, but not to E, F or G.

- Premise 4.1 in Section 4.1.2 is adopted to avoid some operation checking. Since by assumption, the current database state is semantically correct, we would need to check an operation only because the database state transition caused by the operation may violate the SIC. For example, an insertion of an *RX* occurrence may violate *"if E RX F then E RY G"*, but a deletion of a *RX* could not. Similarly, an insertion of an *RX* occurrence may violate *"if* $\exists$ *at_least 3 RX, E RX F then* $\exists$ *RY, E RY G"*, but a deletion of a *RX* could not.
- *Current_time* and some aggregate functions have special monotonicity properties. For example, a deletion of an *E* occurrence may violate *"max(E.A) > Value"*, but not an insertion of an *E* because, by the monotonicity of function *max*, an insertion of a new occurrence would never decrease the aggregate function value.
- Since a primary key is used as a surrogate to represent an entity or a relationship in a traditional database, its update may imply a deletion of an "old" entity or relationship occurrence followed by an insertion of a "new" entity or relationship occurrence. Therefore, sub-SICs related to deletion and insertion should also be asserted on the update of a primary key.

Note that the algorithms have limited capability to identify avoidable operation checking for SICs involving aggregate function. In addition, it is possible to incorporate more consistency and nonredundancy rules for SIC types into the algorithms to reduce the sub-SICs further. However, it can be claimed that the current algorithms correctly find the relevant object and operation types for a single SIC.

3. **Proper Precondition and Predicate Components.** Basically, the algorithms just rewrite the original SIC so that it becomes more precise and suitable for each

sub-SIC. The following are some brief ideas.

- Subscripts and some special predicates (e.g., *rship_occ_part*) are used to make a sub-SIC precise.
- If a sub-SIC is for an attribute on update, in general, its predicate component contains only an assertion on the attribute requiring it to have some special value after update.
- If a sub-SIC is for an entity or relationship and it does not only assert the values of its attributes, in general, the precondition component is important to identify what is the entity or relationship occurrence to be checked.
- If the constraint violation is because an entity occurrence is deleted, in general, because of the key uniqueness, it is impossible to find another occurrence of the same type to avoid the constraint violation (unless the constraint is one asserting aggregate properties). However, it is possible to "reconnect" an occurrence of one entity type with another occurrence of the other entity type. That is, we might find another relationship occurrence to satisfy the SIC unless both sides of the relationship are bound.
- Constraints that assert explicitly the equality of some key attributes of two entity types, e.g., SICs implied by an *ID Dependency*, *is_a* or *component_of* relationships, should be dealt with specially. Because the key attributes (e.g., SIN) of the two entity occurrences (e.g., *Person* and *Employee*) in fact reference the same physical entity occurrence (e.g., the same physical *Person*), the link should be permanent.

4. **Suggested Violation Actions.** As stated, because of the verification problem, the *SIC elicitation subsystem* could only suggest some restricted and predefined

violation actions appropriate to the certainty of the SIC. A propagation action can be automatically suggested by the *SIC elicitation subsystem* only in the case that there are two simple assertions on two objects in the SIC. In that case, a simple propagation action is suggested to regain a consistent database state. If the database designer has set a certainty threshold (say 75%), any uncertain SIC with certainty less than the threshold will be given a "warning" as its violation action automatically.

5. **Given SIC Names.** A SIC name is given automatically after applying the above algorithms and using some predefined SIC types.

Examples Some examples are included in Appendix I for illustration.

General SIC Information After decomposing a general SIC, we may still need to link its decomposed sub-SICs together since the verification for consistency and nonredundancy may still be needed, and furthermore, a general SIC may later be deleted. In addition, for documentation purposes, it may be desirable to keep the original general SIC information, which contains only the certainty factor (if not default "certain"), precondition and predicate components. Therefore, the *SIC elicitation subsystem* would need to link a general SIC and its decomposed sub-SICs together.

7.3.1 Representation of Generic SICs

The principles of the reformulation and decomposition algorithms can be similarly applied to produce the generic sub-SICs for the generic object types *Entity**, *Relationship**, *Entity*.Attribute**, and *Relationship*.Attribute**. However, the SIC representation for

the generic types is complicated because the data dictionary retrieval and manipulation, and the pre-condition contexts must be explicitly stated. The input information and manipulation logical predicates used in generic SICs are listed in Appendix B. This subsection introduces the representation of generic SICs for some common SIC types and discusses the related issues. In all of the following cases, the principle of SIC specialization allows us to omit the explicit SIC representations for specific object types if we have the representations for generic object types.

Domain Constraint Representation — SIC Aggregation and Specialization By the principle of SIC aggregation, domain constraints on insertion of an entity/relationship occurrence can be simulated by applying the domain constraints of all its attributes. There are three separate sub-SICs for restricting not-null, uniqueness, and nonvolatility, respectively, and another sub-SIC dealing with data-type, format, and value. For restricting the insertion of an entity, we need one sub-SIC, which would call the above related sub-SICs (except for the nonvolatility) for asserting attribute domain constraints. In total, the five sub-SICs in Appendix J are sufficient to represent all domain constraints regardless of the number of entity types and their attributes in a database. The number of SICs that must be specified explicitly is dramatically reduced through using the SIC abstraction concepts. Similarly, five sub-SICs are needed for relationship types and their attributes in a database.

Primary Key Constraint Representation — SIC Association and Specialization A number of SICs must be specified to capture the possible inconsistent states of a database when updating a primary key. The SIC association and specialization concepts will be used to reduce the number of explicit SICs required. During the database design

phase, if a SIC is identified for the insertion (or deletion) of a relationship and there should be a sub-SIC for checking the update of a part of its key, its SIC name is added into an associated *SIC_Name_Set* of the affected key attribute. The *SIC_Name_Set*s of key attributes of a relationship type may be different. However, in the case of a SIC that is relevant to the insertion (or deletion) of an entity type, all affected key attributes of the entity have the same *SIC_Name_Set*.

For example, suppose that we have a SIC such as "*if an employee is assigned to a project, he (she) must participate in an insurance plan*", and assume the key of the employee and project are *EmpId* and *ProjId*, respectively. The name of the sub-SIC restricting an insertion of the relationship *Assigned_to* would be inserted into the *SIC_Name_Set* of a special logical predicate (*associated_PKSICs_I*) for the key attribute *Assigned_to.EmpId*. Note that since the non-sharing entities are not concerned, the update of another key attribute *Assigned_to.ProjId* of the relationship is not restricted. Similarly, the name of another sub-SIC restricting a deletion of the relationship *Insure*, is inserted into *Associated_PKSICs_D* for *Insure.EmpId*.

Two "set-SICs" in Appendix J are needed for the key attributes of *Relationship**. (Similarly, there are two "set-SICs" for key attributes of *Entity**.) By the principle of SIC association, the enforcement of such a "set-SIC" is the same as the enforcement of all of its "member-SICs".

Other SIC Representation — SIC Specialization A number of other SIC types[98] can be similarly represented. Usually, these SICs types can be described in the "closed form" of predicates, that is, without further arbitrary restrictions. If a DBMS finds

[98]These SIC types include: **Composite_Attribute_Unique Constraint, Absolute Maximum Cardinality Constraint of an Entity Type,** "traditional" relative cardinality constraints (i.e., **Totality Constraint,** and **Relative Maximum Cardinality Constraint**), **Incidence Constraint,**

the related information on a specific entity, relationship or attribute type, e.g., *symmetric(Married_to)* indicating its symmetry, the related sub-SICs would be automatically inherited from the generic types.

The number of such generic SICs stored in the *SIC maintenance subsystem* would depend on the complexity of the application at hand. It is possible that in a special database application all SICs can be represented as generic SICs in advance so that the actual invocation of the reformulation and decomposition algorithms might be almost totally avoided during the database design consultation session.

Some examples of the above generic SICs are included in Appendix J.

Some Improvements The generic SICs in Appendix J have two problems, which can be solved as follows.

- **Uniform Certainty Factors.** It is assumed that the generic SICs are all "certain" by default. If we consider the possibilities that a few SIC types for some specific object types may be "uncertain", the certainty factors need to be stored in the input predicates for these specific object types.
- **Uniform Violation Actions.** Because of the SIC inheritance principle, specific object types would inherit the same violation actions from the generic object types. The advantage of this is that the *elicitation sub-system* only needs to ask the

---

Symmetry Property of a Relationship, Transitivity Property of a Relationship, Subset_Relationship SIC, Relationships_Union Special SIC, Exclusive Relationship SIC, Exclusive Occurrence SIC, Not_And_Relationships SIC, Either_Existence_Relationships SIC, Relationship_Before_Relationship SIC, Relationship_Not_Before_Relationship SIC, Relationships_Join SIC, Relationships_Depends_on_LoopN_Relationships SIC, Weak_Entity SIC, ID_Dependency_Relationship SIC, Weak_Relationship SIC, Completeness_Mapping SIC, Relationships_Intersection Special SIC, Relationship_Trigger_Relationship SIC, Exclusion between Entity Types, Entities_Intersection Special SIC, and **Entities_Union Special SIC.**

database designer the violation action once for each of these common SIC types. However, this rigidity may not be suitable for all applications. An improvement would be to store a violation action "list" in the input predicate for each related specific object type. The number of elements in the list corresponds to the number of its sub-SICs. Each element is a violation action for each sub-SIC. Then the sub-SICs of a specific object type would not inherit the violation action from its generic object type, but have its own "custom-made" violation action. Considering also the above problem of uniform certainty factors, we may allow both the certainty factor and the violation action in generic SICs to be variables. They would be bound with actual values when a specific object type satisfies the preconditions.

7.4 Transforming SICs to Relational Form

The final result of the consultation is a logical database specification implemented in the relational model. These SICs may be later enforced by an integrity maintenance subsystem in a relational database system. Therefore, the incorporated SICs referencing entities and relationships in the E-R-SIC model must be automatically transformed by the *elicitation subsystem* into SICs referencing the corresponding relations in the relational model.

Relationship Representation in the Relational Model When constructing the relations in the relational model by using the E-R descriptions, each entity type is naturally represented by a separate relation. However, there are two alternatives for representing a binary relationship type in the relational model. Some researchers favour always representing a relationship type by using a separate relation rather than a foreign key because

they argue that foreign keys decrease the adaptability of database designs ([Wilmot, 1984]) and "[E-R consistent relational schema] assures a greater adaptability to changes not concerning the structure of the modeled information, such as the cardinality of relationships" ([Makowsky, et al., 1986, p. 321]). In addition, by adopting the separate relation approach we would have the same SIC representations in both the E-R-SIC model and the relational model.

However, because of access efficiency considerations, the foreign key approach still prevails in practice. This research allows the foreign key approach to representing a relationship type having (1,1) relative cardinalities. The cardinalities stability criterion is adopted if there is a tie to decide which entity key will become the foreign key. Suppose that a relationship type R relates to entity types E and F. If the relationship type R has any attribute, it is represented by a separate relation in the relational model. Otherwise, its representation is based on the following rules[99]:

1. Suppose only one of the involved entity types has (1,1) cardinalities. If E has the (1,1) cardinalities, add the primary key of F to the E relation as a foreign key. Otherwise, add the primary key of E to the F relation as a foreign key.

2. Suppose both entity types, E and F, have (1,1) cardinalities.

   (a) The decision is based on the stability of the cardinalities. If the (1,1) cardinalities of F are more likely to change in the future, add the primary key of F to the E relation as a foreign key. Otherwise, add the primary key of E to the F relation as a foreign key.

[99]There is an exception. If the relationship is a relationship via which an **ID Dependency** happens, an *is_a* or *component_of* relationship, the entity type with the (1,1) relative cardinalities already has the primary key of the other entity type as its candidate key attributes. These key attributes can play both the role of (part of) a candidate key and a foreign key. We need not add the key attributes of the other entity type to it.

(b) If the above cannot be decided, the choice is then based on the relative frequencies of *"F of E"* or *"E of F"* type queries. If the query *"F of E"* would be encountered more often, add the primary key of F to the E relation as a foreign key. Otherwise, add the primary key of E to the F relation as a foreign key.

(c) If neither of the above can be decided, the choice depends on the last resort — how the database designer specifies the relationship, *"E R F"* or *"F R E"*. It is based on the heuristic that future queries may be similar to the way the database designer states the relationship although he (she) may not admit it. If originally the relationship is expressed by the database designer as *"E R F"*, add the primary key of F to the E relation as a foreign key. Otherwise, add the primary key of E to the F relation as a foreign key.

3. In all other cases, construct a separate relation for the relationship.

SIC Representation in the Relational Model Although we allow the foreign key approach to representing relationships in the special cases of (1,1) cardinalities, this kind of representation is only for query and processing efficiency. The semantics should be the same as in the E-R-SIC model. Therefore, the entity and relationship descriptions would still be stored in the data dictionary of a relational database. There would also be the same classification of SIC types. However, the adoption of the foreign key approach would cause some semantic confusion because now a relation could represent an "entity", or "relationship", or even both. In this research, if a relation represents both an entity type and one or more relationship types, it is deemed a special entity type with some information kept in the data dictionary. The associated special information specifies the relationship types hidden in it (by adding the key of the other entity type(s) to it as

a foreign key) and the relative cardinalities of the other entity type(s) in this hidden relationship type[100].

The (1,1) cardinalities of one entity type in a relationship type would cause some SICs to be redundant or inconsistent. In addition, some SICs in the E-R representation need not be transformed when constructing relations because either they do not mention an explicit relationship or the relationship representation is not relevant to them[101]. These SIC's representation in the relational model would be the same as corresponding ones in the E-R-SIC model.

The Algorithms The general transformation algorithms for transforming SICs in the E-R-SIC model into SICs referencing the corresponding relations in the relational model are described in Appendix K. It is relatively simple to transform the relationship name and the manipulation predicates of its participating entities[102]. However, in addition to the primary key update problem, in the relational model we have a problem of the update of foreign key attributes owing to the well-known semantic overload issue. The update of any attributes of a foreign key would imply the deletion of an old relationship occurrence and the insertion of a new one. The SIC association and specialization concepts can be applied here too.

[100]That is, two related *"relationship_participant"* predicates would be deleted and a new *"relationship_hidden_entity"* predicate will created by the *SIC elicitation subsystem.*

[101]Some of these SIC types are: all single entity attribute SICs, all entity SICs, SICs on a single relationship's attributes, **Entities_Intersection Special SICs**, **Entities_Union Special SICs.** In addition, if a relationship is declared to be complete, usually the relative cardinalities of any involved entity type should not be (1,1). Otherwise, the other side of the entity type can only have exactly one occurrence, which is not practical. Therefore, the **Completeness_Mapping SIC** usually need not be transformed.

[102]By taking the foreign key approach to representing relationships, some SICs would become redundant. For example, we do not need a SIC requiring the relative maximum cardinality to be 1 for the "entity" relation in which the relationship is now hidden because the foreign key is single-valued. The algorithms would also remove them.

Examples corresponding to the ones in the E-R-SIC model are included in Appendix L for illustration.

Generic SICs It is also desirable to apply the SIC abstraction concept here to represent generic SICs. In principle, we can apply the above transformation algorithms to transform the generic SICs in the E-R-SIC model into corresponding ones in the relational model although the references to some new information from the data dictionary may be needed. However, the foreign key approach would need additional sets of predefined sub-SICs. Because now relationship representation is not uniform, we need to consider each of the three possible relation representations for each relationship. If a sub-SIC involves two relationships, it would have nine possible combinations of representations; if it involves three relationships, the possible combinations would become twenty-seven although some impossible cases can be excluded[103].

Such an analysis and preparation of pre-defined generic SICs could be carried out for SIC types involving only a few relationships. However, it becomes impossible when a SIC type may involve an unknown number of relationships (e.g., the intersection of relationships). In those SIC types, we would be forced to pre-define generic SICs only for the simple cases having at most three relationships involved.

Some of these SICs representations in the relational model are included in Appendix M for illustration.

[103]For example, suppose that we have an **Exclusive_Relationship SIC**, such as $RX \| RY$ *relative to* E where RX type connects the entity types E with F, and RY connects the entity types E with G. We would know that E cannot have (1,1) cardinalities in either RX or RY. So we need not consider those impossible representations for relationships RX and RY — adding the primary key of F or G as the foreign entity of E. However, F may have (1,1) cardinalities in RX; and G may have (1,1) cardinalities in RY too. So we would need two more sets of representations for the predefined sub-SICs in addition to the original one in the E-R-SIC model.

Chapter 8

Conclusions and Further Research

8.1 Conclusions and Contributions

Conclusions This research has presented two models. The E-R-SIC model is a comprehensive modelling tool for helping the database designer systematically incorporate semantic integrity constraints that are relevant to attributes, entities, and relationships in a database. The SIC Representation model is used to represent precisely the features of these SICs. The declarative and operational semantics of each SIC are specified. By using these two models, the data integrity semantic constraints on the allowable states and state transitions can be completely modelled and properly represented in a database schema. The focus of database designers will be changed from traditionally emphasizing only structure, functional dependencies, efficiency, etc. to describing **data semantics**. In a database, the number of explicit SICs specified using the Representation model would not be huge because of the application of SIC abstractions and the representation of generic SICs. These SICs could be efficiently enforced because of the characteristics of the SIC Representation model.

Both the Representation model and the E-R-SIC model are application-domain independent. They are suitable for implementation as part of an automated database design

system. Conceptually, this research proposes a *SIC elicitation subsystem.* The *SIC elicitation subsystem* would detect where general SICs may be needed and prompt a database designer to confirm or provide them. The subsystem would automatically decide for what data objects and on what database operations these SICs should be enforced, reformulate them (decompose them into sub-SICs if necessary) in terms of the Representation model, and suggest some violation actions. The subsystem would have some built-in consistency and nonredundancy rules for different SIC types, would verify the consistency and nonredundancy of SICs to some extent and transform them into corresponding ones in the relational model. This kind of automated database design system would provide more assistance to a database designer in modelling the data semantics.

Contributions Most previous SIC research concentrates on classifying and efficiently enforcing a few types of SICs. Current languages do not represent all features of a SIC precisely. Existing automated database aids do not provide adequate facilities for incorporating SICs into a design. This research contributes to our understanding of database semantic integrity. On the theoretical side, there are the following contributions:

1. The SIC Representation model is defined to represent precisely the necessary features for specifying declarative and operational semantics of a SIC.

2. The E-R-SIC model is proposed to incorporate dynamic and static SICs in a database schema rather than in transactions and programs.

3. Algorithms are provided to reformulate and decompose SICs elicited using the E-R-SIC model, and to transform them into corresponding ones in the relational data model.

On the practical side, there are the following contributions:

1. A *SIC elicitation subsystem* has been proposed to help the database designer design SICs in addition to the structure part of a database schema.

2. This research provides a foundation for overcoming the well-known problem of representing data integrity semantics in current relational database systems. The resulting database would have the advantages of embedded SICs as described in Chapter 1. The SIC representation would facilitate the efficient enforcement.

3. Although not empirically tested, we may conjecture that the information system conceptual design would more completely and systematically include data and information system semantics. The database designer would be able to model more data integrity semantics, thereby reducing the need for this information to be included in application programs. This research provides a starting point for future empirical tests.

8.2 Future Research Extensions to this Dissertation

This dissertation is part of a research program at the University of British Columbia to formalize the database design process and make databases more intelligent. There are many areas for further research to extend this dissertation. Some of these are suggested below.

- **Non-binary Relationships.** Currently, the E-R-SIC model assumes all relationship types to be binary. It is assumed that a database designer would know how

to use binary relationships to simulate non-binary ones (including recursive relationships, ternary relationships and relationships of higher degree). This restriction could be relaxed to allow the database designer to model non-binary relationships directly.

- **Development and Implementation of Efficient Algorithms to Assure the Consistency, Nonredundancy, and Even Optimization of SICs.** To prove the consistency, nonredundancy and optimization of arbitrary SIC statements is still an open issue. In addition, to design and implement those verification algorithms needs more research. These are challenges for researchers in the fields of management information system, computer science, and mathematics.

- **Integration of SICs Elicited from Multiple Sources.** This dissertation is based on a simplified assumption of a single database designer. *View integration or synthesis* is an important topic in database design research. Previous research (e.g., [Wagner, 1989]) have addressed this issue without considering SIC specifications. It is also possible that the SICs need to be obtained from several database designers. Future researchers may address this complicated issue.

- **Programming Implementation.** Future research would need to implement the proposed *SIC elicitation subsystem.* Then the produced *SIC elicitation subsystem* must be merged with a system to construct the structure of a schema (e.g., the *View Creation System* [Storey, 1988]) to provide the complete database schema design assistance to a database designer. It is also desirable to incorporate it with a view integration system (e.g., *AVIS* [Wagner, 1989]) and a future SIC integration system in the case that multiple sources for a database specification are needed. A complete automated database design system needs to be implemented.

- **Add-on Knowledge, Capabilities and Features of the SIC Elicitation Subsystem.** Based on the E-R-SIC model proposed in this research, future researchers may add some common-sense, domain-dependent, and organization-dependent knowledge to the proposed automated database design system to make it more intelligent and provide more efficient and effective assistance for a database designer. Ideally, the system may have a learning capability so that it can accumulate knowledge each time it is used. It is also desirable to provide some natural language facilities and graphical interface so that a database designer can more easily describe entities and relationships, and directly design an E-R database on the screen.

- **Transaction Modelling.** In order to capture transaction-driven semantics and check the SICs related to transactions more efficiently, specifications of transactions, i.e., user-predefined operations, need to be specified. Future research may propose algorithms to transform SICs identified by the E-R-SIC model to the pre-conditions and post-conditions of transactions.

- **Application of SICs in an Expert Database System.** As stated earlier, SICs can be used to facilitate intelligent query evaluation and provide deductive capabilities. Further research may explore how to include and apply SICs represented by the Representation model in an expert database system.

- **Design and Management of an Integrity Maintenance Subsystem in the Relational Database.** Very few kinds of SICs are enforced in commercial relational database systems. One reason for this is probably concern for efficiency. Based on SIC specifications identified in this research, future researchers may design an integrity maintenance subsystem to enforce SICs efficiently in a relational database. The management of SICs, i.e., the insertion or deletion of SICs, after the database is populated should be carefully taken into consideration. Some

administrative procedures may need to be invented to handle the "change of SICs".

- **Empirical Research for Testing "Usefulness".** Future empirical researchers may test two kinds of usefulness. The first is the usefulness of using database embedded SICs versus the traditional approach of enforcing integrity via application software. The second is the usefulness of the design approach adopting an automated database design system compared with the manual design approach (without any assistance of an automated design system) to incorporate the necessary SICs for embedding in a database. The automated database design system proposed in this dissertation can be taken as a tool.

Bibliography

[1] Abiteboul, S., and Vianu, V., "Transactions and Integrity Constraints", *Proc. of the Second ACM SIGACT-SIGMOD Symposium of Principles of Database Systems*, Portland, Oregon, May 1985, pp. 193-204.

[2] Aho, A. V., Hopcroft, J. E., and Ullman, J. D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

[3] Arisawa, H., and Miura, T., "On the Properties of Extended Inclusion Dependencies", *Proc. of the Twelfth International Conference on Very Large Data Base*, Kyoto, August 1986, pp. 449-457.

[4] Azar, N., and Pichat, E., "Translation of an Extended Entity-Relationship Model into the Universal Relation with Inclusion Formalism", in *Entity-Relationship* (the Fifth International Conference on E-R Approach, France, 1986) edited by S. Spaccapietra, Elsevier Science Publishers B.V. (North-Holland), 1987, pp. 253-268.

[5] Benci, E., Bodart, F., Bogaert, H., and Cabanes, A., "Concepts for the Design of a Conceptual Schema", in *Modelling in Data Base Management Systems*, edited by G.M. Nijssen, North-Holland Publishing Co., 1976, pp. 181-200.

[6] Bernstein, P. A., Blaustein, B. T., and Clarke, E. M., "Fast Maintenance of Semantic Integrity Assertions Using Redundant Aggregate Data", *Proc. of the Sixth International Conference on Very Large Data Base*, 1980, pp. 126-136.

[7] Bertino, E., and Apuzzo, D., "Integrity Aspects in Data Base Management Systems", *Proc. Trends & Applications*, Gaithereshurg, Maryland, 1984, pp. 43-52.

[8] Biller, H., and Neuhold, E. J., "Semantics of Data Bases: The Semantics of data Models", *Information System*, Vol. 1 No. 3, 1978, pp. 273-292.

[9] Borgida, A., "Language Features for Flexible Handling of Exceptions in Information Systems", *ACM Transactions on Database Systems*, Vol. 10, No. 4, December 1985, pp. 565-603.

[10] Bouzeghoub, M., and Gardarin, G., "The Design of An Expert System for Database Design" in *New Applications of Data Bases*, edited by G. Gardarin and E. Gelenbe, 1984, pp. 203-223.

[11] Bouzeghoub, M., Gardarin, G., and Metais, E., "Database Design Tools: An Expert System Approach:", *Proc. of the Eleventh International Conference on Very Large Data Base*, Stockholm, August 1985, pp. 82-95.

[12] Bouzeghoub, M., and Metais, E., "SECSI: An Expert System Approach for Database Design", in *Information Processing 86*, edited by H.J. Kugler, Elsevier Science Publishers B.V. (North-Holland), 1986, pp. 251-257.

[13] Bracchi, G., Furtado, A., and Pelagatti, G., "Constraints Specification in Evolutionary Data Base Design", in *Formal Models and Practical Tools for Information Systems Design*, edited by H.-J. Schneider, North-Holland Publishing Co., 1979, pp. 149-165.

[14] Brady, L. I., and Dompney, C. N. G., "Dynamics of Database Semantic Integrity or Managing the Meaning of Data", *Proc. of the Australian Computer Conference*, Sydney, November 1984, pp. 82-96.

[15] Brägger, R. P., Dudler, A., Rebsamen, J., and Zehnder, C. A., "Gambit: An Interactive Database Design Tool for Data Structures, Integrity Constraints and Transactions", *International Conference on Data Engineering*, Los Angeles, California, August 1984, pp. 399-407.

[16] Brodie, M. L., *Specification and Verification of Database Semantic Integrity*, Ph.D. Thesis, University of Toronto, 1978 (also as Technical Report CSRG-91, April 1978).

[17] Brodie, M. L., "Association: A Database Abstraction for Semantic Modelling", in *Entity-Relationship Approach to Information Modelling and Analysis* (the Second International Conference on E-R Approach, 1981), edited by P. P. Chen, Elsevier Science B.V. (North-Holland), 1983, pp. 577-602.

[18] Brodie, M. L., "On the Development of Data Models", in *On Conceptual Modelling*, edited by M.L. Brodie, J. Mylopoulus and J.W. Schmidt, Spring-Verlag, 1984, pp. 19-47.

[19] Brodie, M. L., "Database Management: A Survey", in *On Knowledge-Base Management Systems*, edited by M.L. Brodie and J. Mylopoulus, Springer-Verlag, 1986, pp. 201-218.

[20] Brodie, M. L., and Manola, F., "Database Management: A Survey", in *Foundations of Knowledge Base Management: Contributions from Logic, Databases, and Artificial Intelligence Applications*, edited by J. W. Schmidt and C. Thanos, Springer-Verlag, 1989, pp. 205-234.

[21] Brodie, M. L., and Ridjanovic, D., "On the Design and Specification of Database Transactions", in *On Conceptual Modelling*, Edited by M.L. Brodie, J. Mylopoulos, and J. W. Schmidt, Spring-Verlag, 1984, pp. 277-306.

[22] Bry, F., and Manthey, R., "Checking Consistency of Database Constraints: a Logical Basis", *Proc. of the Twelfth International Conference on Very Large Data Base*, Kyoto, August 1986, pp. 13-20.

[23] Bubenko, J. A., "Information Modeling in the Context of System Development", *Information Processing 80*, edited by S. H. Lavington, North-Holland Publishing Co., 1980, pp. 395-411.

[24] Carsnell, J. L., and Navathe, B. "SA-ER: A Methodology that Links Structured Analysis and Entity-Relationship Modelling for Database Design", in *Entity-Relationship* (the Fifth International Conference on E-R Approach, France, 1986) edited by S. Spaccapietra, Elsevier Science Publishers B.V. (North-Holland), 1987, pp. 381-397.

[25] Casanova, M. A., and Furtado, A. L., "On the Description of Database Transition Constraints Using Temporal Languages", in *Advances in Database Theory, Vol. 2,* edited by H. Gallaire, J. Minker and J.M. Nicolas, Plenum Press, New York, 1984, pp. 211-236.

[26] Casanova, M. A., and Tucherman, L., "Enforcing Inclusion Dependencies and Referential Integrity", *Proc. of the Fourteenth International Conference on Very Large Data Base*, California, 1988, pp. 38-49.

[27] Casanova, M. A., and Vidal, V. M. P., "Towards a Sound View Integration", *Proc. of the Second ACM SIGACT-SIGMOD Symposium of Principles of Database Systems*, Atlanta, George, March 1983, pp. 36-46.

[28] Cauvet, C., Proix, C., and Rolland C., "A Knowledge Base for an Information System Design Tool", in *Methodologies for Intelligent Systems*, edited by Z.W. Ras, and M. Zemankova, Elsevier Science Publishing Co. Inc., 1987, pp. 56-63.

[29] Ceri, S., and Widom, J., "Deriving Production Rules for Constraint Maintenance", IBM Research Report, RJ 7348, March 1, 1990. (A short version appears in *Proc. of the 16th VLDB Conference*, Australia, 1990, pp. 566-577).

[30] Chakravarthy, U. S., Minker, J., and Grant, J., "Semantic Query Optimization: Additional Constraints and Control Strategies", in *Expert Database Systems*, edited by L. Kerschberg, Benjamin/Cummings Pub. Co. Inc., 1987, pp. 345-377.

[31] Chen, P. P., "The Entity-Relationship Model – Toward a Unified View of Data", *ACM Transactions on Database Systems*, Vol. 1 No. 1, December 1976, pp. 9-36.

[32] Chen, P. P., "Database Design Based on Entity and Relationship", in *Principles of Database Design, Vol. 1: Logical Organization*, edited by S.B. Yao, Prentice-Hall, 1985, pp. 174-210.

[33] Choobineh, J., "Form Driven Conceptual Data Modelling", Ph.D. Dissertation, Dept. Management Information Systems, University of Arizona, 1985.

[34] Choobineh, J., Mannino, M. V., Nunamaker, J.F., and Konsynski, B.R., "An Expert Database Design System Based on Analysis of Forms", *IEEE Transaction on Software Engineering*, Vol. 14, No. 2, February 1988, pp. 242-253.

[35] Codd, E. F., "Extending the Database Relational Model to Capture More Meaning", *ACM Transactions on Database Systems*, Vol. 4, No. 4, December 1979, pp. 397-434.

[36] Cohen, J., "Constraint Logic Programming Languages", *Communications of the ACM*, Vol. 33, No. 7, July 1990, pp. 52-68.

[37] Colmerauer, A., "An Introduction to Prolog III", *Communications of the ACM*, Vol. 33, No. 7, July 1990, pp. 69-90.

[38] Cosmadakis, S. C., and Kanellakis, P. C., "Equation Theories and Database Constraints", *Proc. of the 17th Annual ACM symposium on Theory of Computing*, Rhode Island, May 1985, pp. 273-284.

[39] Dampney, C. N. G., "Specifying a Semantically Adequate Structure for information Systems and Databases", in *Entity-Relationship Approach* (the Sixth International Conference on E-R Approach, New York , Nov. 1987), edited by S. T. March, Elsevier Science Publishers, B.V. (North-Holland), 1988, pp. 165-188.

[40] Date, C. J., *An Introduction to Database System, Vol. II*, Addison-Wesley Publishing Co., Reading, Massachusetts, 1983.

[41] Date, C. J., *A Guide to the SQL Standard*, Addison-Wesley Publishing Co., Reading, Massachusetts, 1987.

[42] Davis, J. P., and Bonnell, R. D., "Modelling Semantic Constraints with Logic in the EARL Data Model", *The Fifth International Conference on Data Engineering*, Los Angeles, California, February 1989, pp. 226-233.

[43] de Castilho, J. M. V., Casanova, M. A., and Furtado, A. L., "A Temporal Framework for Database Specifications", *Proc. the Eighth International Conference on Very Large Data Base*, Mexico City, Mexico, 1982, pp. 280-291.

[44] Delobel, C., "Normalization and Hierarchical Dependencies in the Relational Data Model", *ACM Transactions on Database Systems*, Vol. 3, No. 3, September 1978, pp. 201-222.

[45] Dogac, A., and Chen P. P., "Entity-Relationship Model in the ANSI/SPARC Framework", in *Entity-Relationship Approach to Information Modelling and Analysis* (the second International Conference on E-R Approach, 1981), edited by P.P. Chen, Elsevier Science B.V. (North-Holland), 1983, pp. 357-374.

[46] Dogac, A., Chen P. P. and Erol, N., "the Design and Implementation of an Integrity Subsystem for the Relational DBMS RAP", in *the Fourth Conference on Entity-Relationship Approach*, Chicago, Illinois, October 1985, pp. 295-302.

[47] dos Santos, C. S., Neuhold, E. J., and Furtado, A. L., "A Data Type Approach to the Entity-Relationship Model", in *Entity-Relationship Approach to System Analysis and Design* (the First International Conference on E-R Approach), edited by P. P. Chen, North-Holland Publishing Co., 1980, pp. 103-119.

[48] Ehrich, H. D., Lipeck, U. W., and Gogolla, M., "Specification, Semantics, and Enforcement of Dynamic Database Constraints", *Proc. of the Tenth International Conference on Very Large Data Base*, Singapore, August 1984, pp. 301-308.

[49] Eswarn, K. P., and Chamberlin, D. D., "Functional Specifications of a Subsystem for Data Base Integrity", *Proc. the Second International Conference on Very Large Data Base*, Framingham, Massachusetts, September 1975, pp. 48-68.

[50] Etzion, O., "PARDES – a Model for Supporting Derivation Closure", Working Paper, Computer and Information Sciences Department, Temple University, Philadelphia, Pennsylvania,1989.

[51] Fernandez, E. B., Summers, R. C., and Wood, C., *Database Security and Integrity*, Addison-Wesley Publishing Co., Reading, Massachusetts, 1981.

[52] Fleming, C. C., and Halle, B. V.,*Handbook of Relational Database Design*, Addison-Wesley Publishing Co., Reading, Massachusetts, 1989.

[53] Fong, E., and Kimbleton, S. R., "Database Semantic Integrity for a Network Data Manger", *AFIPS Proceedings of National Computer Conference*, (Vol. 49), California, May 1980, pp. 261-268.

[54] Frost, R. A. (ed).,*Database Management Systems*, Granada Publishing Ltd., London, 1984.

[55] Furtado, A. L., dos Santos, C. S., and de Castilho, J. M. V., "Dynamic Modelling of a Simple Existence Constraint", *Information Systems*, Vol. 6, 1981, pp. 73-81.

[56] Furtado, A. L., and Neuhold, E. J., *Formal Techniques for Data Bases Design*, Springer-Verlag, Berlin, 1986.

[57] Furtado, A. L., Casanova, M. A., and Tucherman, L., "The CHRIS CONSULTANT", in *Entity-Relationship Approach* (the Sixth International Conference on E-R Approach, New York , Nov. 1987), edited by S. T. March, Elsevier Science Publishers, B.V. (North-Holland), 1988, pp. 515-532.

[58] Gardarin, G., and Melkanoff, M., "Proving Consistency of Database Transactions", *Proc. of the Fifth International Conference on Very Large Data Base*, Brazil, October 1979, pp. 291-298.

[59] Goldstein, R. C., *Database: Technology and Management*, John Wiley & Sons, 1985.

[60] Goldstein, R. C., and Storey, V. C., "Data Abstraction: The Impact on Database Management", Working Paper, Faculty of Commerce and Business Administration, The University of British Columbia, October 1990.

[61] Goldstein, R. C., and Wagner, C., *Manual of Instruction Database Software for Microcomputers*, Faculty of Commerce and Business Administration, The University of British Columbia, December 1988.

[62] Hammer, M. M., and McLeod, D. J., "Semantic Integrity in a Relational Data Base System", *Proc. the Second International Conference on Very Large Data Base*, Framingham, Massachusetts, September 1975, pp. 25-47.

[63] Hammer, M. M., and McLeod, D. J., "A Framework for Data Base Semantic Integrity", *Proc. the Second International Conference on Software Engineering*, San Franciso, California, October 1976, pp. 498-504.

[64] Hammer, M. M., and McLeod, D. J., "Database Description with SDM: A Semantic Database Model", *ACM Transaction on Database Systems*, Vol. 6, No. 3, September 1981, pp. 351-386.

[65] Hentenryck, P. V., *Constraint Satisfaction in Logic Programming*, MIT Press, Massachusetts, 1989.

[66] Hentenryck, P. V., "Incremental Constraint Satisfaction in Logic Programming", in *Logic Programming: Proc. of the Seventh International Conference*, edited by D. H. D. Warren and P. Szeredi, 1990, pp. 189-202.

[67] Henschen, l. J., McCune, W. W., and Naqvi, S. A., "Compiling Constraint-Checking Programs from First-Order Formulas", in *Advances in Database Theory, Vol. 2,* edited by H. Gallaire, J. Minker and J. M. Nicolas, Plenum Press, New York, 1984, pp. 145-169.

[68] Heuser, C. A., and Richter, G., "On the Relationship between Conceptual Schema and Integrity Constraints on Databases", in *Database Semantics (DS-1)*, edited by T. B. Steel, Jr. and R. Meersman, Elsevier Science Publishers, B.V. (North-Holland), 1986, pp. 27-39.

[69] Hillier, F. S., and Lieberman, G. J., *Introduction to Operations Research*, the fourth edition, Holden-Day, 1986.

[70] Ho, H. C., "Integrity Control in a Relational Database", Technical Report S.O.C.S.828, School of Computer Science, McGill University, Montreal, Canada, March 1982.

[71] Hohenstein, U. and Hülsmann, K., "A Language for Specifying Static and Dynamic Integrity Constraints", in *the Tenth International Conference on E-R Approach*, San Mateo, California, October, 1991, (proceedings edited by T.J. Teorey), pp. 389-416.

[72] Holsapple, C., Shen, S., and Whinston, A., "A Consulting System for Database Design", *Information System*, Vol. 7, No. 3., 1982, pp. 281-296.

[73] Hsu, A., and Imielinski, T., "Integrity Checking for Multiple Updates", *Proc. of ACM SIGMOD International Management of Data*, Austin, Texas, May 1985, pp. 152-167.

[74] Hsu, C., Perry, A., Bouziane, M. and Cheung, W., "TSER: A Data Modelling System using the Two-Stage Entity-Relationship Approach",in *Entity-Relationship Approach* (the Sixth International Conference on E-R Approach, New York, November 1987), edited by S. T. March, Elsevier Science Publishers, B.V. (North-Holland), 1988, pp. 497-514.

[75] Hull, R., and King, R., "Semantic Database Modelling: Survey, Applications, and Research Issues", *ACM Computing Surveys*, Vol. 19, No. 3, September 1987, pp. 201-260.

[76] Jaffar, J., and Lassez, J-L., "Constraint Logic Programming", *14th Annual SCM Symposium on Principles of Programming Languages*, Munich, West Germany, January 1987, pp. 111-119.

[77] Jaffar, J., and Michaylov, S., "Methodology and Implementation of a CLP system", *Proc. of 4th International Conference on Logic Programming*, edited by J-L. Lassez, MIT Press, 1987, pp. 196-217.

[78] Jarke, M., and Vassiliou, Y., "Databases and Expert Systems: Opportunities and Architectures for Integration", in *New Applications of Databases*, edited by G. Gardarin and E. Gelenbe, Academic Press London, 1984, pp. 185-201.

[79] Jones, C. B., *Systematic Software Development Using VDM*, Prentice-Hall, 1986.

[80] Kawaguchi, A., Taoka, N., Mizoguchi, R., Yamaguchi, T., and Kakusho, O., "An Intelligent Interview System for Conceptual Design of Database", *Proc. ECAI* 1986.

[81] Kennedy, A. J., and Yen, D. C., "Enhancing a DBMS Through the Use of an Expert System", *Journal of Information Management*, Spring 1990, pp. 55-61.

[82] Kent, W., "Entities and Relationships in Information", in *Architecture and Models in Data Base Management Systems*, edited by G.M. Nijssen, North-Holland Publishing Co., 1977, pp. 67-91.

[83] Kent, W., "Limitations of Record-Based Information Models", *ACM Transactions on Database Systems*, Vol. 4, No. 1, March 1979, pp. 107-131.

[84] Kent, W., "A Single Guide to Five Normal Forms in Relational Database Theory", *Communications of the ACM*, Volume 26, No. 2, February 1983, pp. 120-125.

[85] Kerstern, M. L., Weigand, H., Dignum, F., Boom, J., " A Conceptual Modelling Expert System", in *Entity-Relationship* (the Fifth International Conference on E-R Approach, France, 1986) edited by S. Spaccapietra, Elsevier Science Publishers B.V. (North-Holland), 1987, pp. 35-48.

[86] Kim, M.-J., Lee, W.-U., and Derniame, J.-C., "Automatic Relational Data Base Designs by Transformation of the Entity-Relationship Model", *IEEE the Second International Conference on Computer and Application*, Beijing, China, 1987, pp. 418-425.

[87] Knuth, E., Hannák, L., Radó, P., "A Taxonomy of Conceptual Foundations", in *Data and Knowledge (DS-2)*, edited by R.A. Meersman and A.C. Sernadas, Elsevier Science Publishers, B.V. (North-Holland), 1988, pp. 205-219.

[88] Kobayashi, I., "Validating Database Updates", *Information Systems*, Vol. 9, No. 1., 1984, pp. 1-17.

[89] Kozaczynski, W., and Lilien, L., "An Extended Entity-Relationship (E^2R) Database Specification and its Automatic Verification and Transformation into the Logical Relational Design", in *Entity-Relationship Approach* (the Sixth International Conference on E-R Approach, New York , November, 1987), edited by S. T. March, Elsevier Science Publishers, B.V. (North-Holland), 1988, pp. 533-549.

[90] Kung, C. H., "A Temporal Framework for Database Specification and Verification", *Proc. of the Tenth International Conference on Very Large Data Base*, Singapore, August 1984, pp. 91-99.

[91] Kung, C. H., "A Tableaux Approach for Consistency Checking", *Information Systems: Theoretical and Formal Aspects*, edited by A. Sernadas, J. Bubenko, Jr., and A. Olivé, 1985, pp. 191-207.

[92] Lafue, G., "Semantic Integrity Dependencies and Delayed Integrity Checking", *Proc. the Eighth International Conference on Very Large Data Base*, Mexico City, Mexico, 1982, pp. 292-299.

[93] Lassez, C., "Constraint Logic Programming", *BYTE*, August 1987, pp. 171-176.

[94] Lauriere, J. L., "A Language and a Program for Stating and Solving Combinatorial Problems", *Artificial Intelligence*, Vol. 10, No. 1, 1978, pp. 29-127.

[95] Lee, R. M., "Logic, Semantics and Data Modelling: An Ontology", in *Data and Knowledge (DS-2)*, edited by R. A. Meersman and A. C. Sernadas, Elsevier Science Publishers, B.V. (North-Holland), 1988, pp. 221-243.

[96] Lee, K., and Lee, S., "An Object-Oriented Approach to Data/Knowledge Modelling Based on Logic", *The Sixth International Conference on Data Engineering*, California, February 1990, pp. 289-294.

[97] Lenzerini, M., and Nobili, P., "On the Satisfiability of Dependency Constraints in Entity-Relationship Schema", *Proc. the Thirtieth International Conference on Very Large Data Base*, Brighton, 1987, pp. 147-154.

[98] Lenzerini, M., and Santucci, G., "Cardinality Constraints in the Entity-Relationship Model", in *Entity-Relationship Approach to Software Engineering* (the Third International Conference on E-R Approach, California, 1983), edited by C. G. Davis, S. Jajodia, P. A. Ng and R. T. Yeh, Elsevier Science Publishers B. V. (North-Holland), 1983, pp. 529-549.

[99] Leveson, N. G., Wasserman, A. I., and Berry, D. M., "BASIS: A Behavioral Approach to the Specification of Information Systems", *Information Systems*, Vol. 8, No. 1, 1983, pp. 15-23.

[100] Ling, T.-W., "Integrity Constraint Checking in Deductive Database using the Prolog not-Predicate", Tech. Report, DISCS Pub. No. NOTRA7/86, National University of Singapore, July 1986.

[101] Ling, T.-W., and Rajagopalan, P., "A Method to Eliminate Avoidable Checking of Integrity Constraints", *Proc. Trends & Applications*, Gaithereshurg, Maryland, 1984, pp. 60-68.

[102] Lipeck, U. W., "Stepwise Specification of Dynamic Database Behaviour", *International Conference on Data Engineering*, Washington, D.C., May 1986, pp. 387-397.

[103] Lockemann, P. C., " Object-Oriented Information Management", *Decision Support System*, 5, 1989, pp.79-102.

[104] Mackworth, A. K., "Consistency in Networks of Relations", *Artificial Intelligence*, Vol. 8, 1977, pp. 99-118.

[105] Mackworth, A. K., and Freuder, E. C., "The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems", *Artificial Intelligence*, Vol. 25, No. 1, January 1985, pp. 65-74.

[106] Mackworth, A. K., "Constraint Satisfaction", *Encyclopedia of Artificial Intelligence*, edited by S. C. Shapiro, J. Wiley & Sons, N.Y., 1987, pp. 205-211.

[107] MaFadden, F. R., and Hoffer, J. A., *Data Base Management*, the second edition, Benjamin/Cummings Publishing Co. Inc., 1988.

[108] Makowsky, J. A., Markowitz, V. M., Rotics, N., "Entity-Relationship Consistency For Relational Schemas", *Proc. International Conference on Database Theory, (Lecture Note in Computer Science V. 243)* edited by G. Ausiello, and P. Atzeni, Springer-Verlag, 1986, pp. 306-322.

[109] Mannila, H., and Räihä, K-J, "Inclusion Dependencies in Database Design", *The Second International Conference on Data Engineering*, Los Angeles, California, February 1986, pp. 713-718.

[110] Maryanski, F., Francis, S., Hong, S., and Peckham, J., "Generation of Conceptual Data Models", Working Paper, Computer Science and Engineering Department, University of Connecticut, 1984.

[111] Maryanski, F., and Hong, S., "A Tool for Generating Semantic Database Applications", *IEEE COMPSAC*, Chicago, Illinois, October 1985, pp. 368-375.

[112] Meersman, R., "Towards Models for Practical Reasoning about Conceptual Database Design", in *Data and Knowledge (DS-2)*, edited by R. A. Meersman and A. C. Sernadas, Elsevier Science Publishers, B.V. (North-Holland), 1988, pp. 245-263.

[113] Mees, M., and Put, F., " Extending a Dynamic Modelling Methods using Data Modelling Capabilities: The Case of JSD", in *Entity-Relationship* (the Fifth International Conference on E-R Approach, France, 1986) edited by S. Spaccapietra, Elsevier Science Publishers B.V. (North-Holland), 1987, pp. 399-418.

[114] Missikoff, M., and Wiederhold, G., "Toward a Unified Approach for Expert and Database Systems", in *Expert Database Systems (Proceedings from the first International Workshop)*, edited by L. Kerschberg, Benjamin/Cummings Publishing, 1986, pp-383-399.

[115] Morgenstern, M., "Active Databases as a Paradigm for Enhanced Computing Environments", *Proc. the Ninth International Conference on Very Large Data Base*, Italy, 1983, pp. 34-42.

[116] Morgenstern, M., "CONSTRAINT EQUATIONS: Declarative Expression of Constraints with Automatic Enforcement", *Proc. the Tenth International Conference on Very Large Data Base*, Singapore, August 1984a, pp. 291-300.

[117] Morgenstern, M., "A Concise Compatible Representation for Quantified Constraints in Semantic Networks", *AAAI-84, Proc. of National Conference on Artificial Intelligence*, Texas, August 1984b, pp. 255-259.

[118] Morgenstern, M., "The Role of Constraints in Databases, Expert Systems, and Knowledge Representation", in *Expert Database Systems*, edited by L. Kerschberg, Benjamin/Cummings Publishing Co., 1986, pp. 351-368.

[119] Morgenstern, M., Borgida, A., Lassez, C., Maier, D., and Wiederhold, G., "Constraint-Based Systems: Knowledge about Data", in *Expert Database Systems* (Proc. from the Second International Conference on EDS), edited by L. Kerschberg, Benjamin/Cummings Publishing Co., 1989, pp. 23-43.

[120] Nakano, R., "Integrity Checking in a Logic-Oriented ER Model", in *Entity-Relationship Approach to Software Engineering* (the Third International Conference on E-R Approach, California, 1983), edited by C. G. Davis, S. Jajodia, P. A. Ng and R. T. Yeh, Elsevier Science Publishers B. V. (North-Holland), 1983, pp. 551-564.

[121] Newell, A., and Simon H. A., "Computer Science as Empirical Inquiry: Symbols and Search", Communications of the ACM, Volume 19, March 1976, pp. 113-126.

[122] Nicolas, J.-M., "Logic for Improving Integrity Checking in Relational Data Bases", *Acta Informatica*, 18, 1982, pp. 227-253.

[123] Nilsson, N. J., *Principles of Artificial Intelligence*, Tioga Publishing Company, 1980.

[124] Obretenov, D., Angelov, Z., Mihaylov, J., Dishlieva, P., and Kirova, N., "A Knowledge-Based Approach to Relational Database Design", *Data & Knowledge Engineering*, 3, 1988, pp. 173-180.

[125] Oren, O., "Integrity Constraints in the Conceptual Schema Language SYSDOC", *the Fourth Conference on Entity-Relationship Approach*, Chicago, Illinois, October 1985, pp. 288-294.

[126] Palmer, I. R., "Practicalities in Applying a Formal Methodology to Data Analysis", in *Data Base Design Techniques I: Requirements and Logical Structures*, edited by S. B. Yao, et al., Spring-Verlag, Berlin, 1982, pp. 147-171.

[127] Papadimitriou, C. H., and Steiglitz, K., *Combinatorial Optimization: Algorithms and Complexity*, Prentice-hall, N.J., 1982.

[128] Paulson, D., *Reasoning Tools to Support System Analysis and Design*, Unpublished Ph.D. Dissertation, The University of British Columbia, Vancouver, B.C., Canada, 1989.

[129] Peckham, J., and Maryanski, F., "Semantic Data Models", *ACM Computing Surveys*, Vol. 20, No. 3, September 1988, pp. 153-189.

[130] Potter, W. D., and Kerschberg, L., " A Unified Approach to Modelling Knowledge and Data", in *Data and Knowledge (DS-2)*, edited by R.A. Meersman and A.C. Sernadas, Elsevier Science Publishers, B.V. (North-Holland), 1988, pp. 265-291.

[131] Proix, C., and Rolland, C., "A Knowledge Base for Information System Design", in *Data and Knowledge (DS-2)*, edited by R.A. Meersman and A.C. Sernadas, Elsevier Science Publishers, B.V. (North-Holland), 1988, pp. 293-306.

[132] Qian, X., and Wiederhold, G., "Knowledge-based Integrity Constraint Validation", *Proceedings of the Twelfth International Conference on Very Large Data Base*, Kyoto, 1986, pp. 3-12.

[133] Qian, X., and Smith, D. R., "Integrity Constraint Reformulation for Efficient Validation", *Proceedings of the thirteenth International Conference on Very Large Data Base*, Brighton, 1987, pp. 417-425.

[134] Raju, K., and Majumdar, A. K., " Fuzzy Functional Dependencies and Lossless Join Decomposition of Fuzzy Relational Database Systems", *ACM Transactions on Database Systems*, Vol. 13, No. 2, June 1988, pp. 129-166.

[135] Ram, S., "Automated Tools for Database Design: Sate of the Art", Working Paper, Dept. of Management Information Systems, College of Business and Public Administration, University of Arizona, 1989.

[136] Reiter, R., " On the Integrity of Typed First Order Data Bases", in *Advances in Database Theory, Vol. 1*, edited by H. Gallaire, J. Minker and J. M. Nicolas, Plenum Press, New York, 1984, pp. 137-157.

[137] Reiter, R., "On Integrity Constraints", *Proc. of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, 1988, pp. 97-111.

[138] Rolland, C., and Proix, C., "An Expert System Approach to Information System Design", in *Information Processing 86*, edited by H. J. Kugler, Elsevier Science Publishers B.V. (North-Holland), 1986, pp. 241-250.

[139] Sakai, H., "On the Optimization of an Entity-Relationship Model", *3rd USA-JAPAN Computer Conference*, San Francisco, California, October 1978, pp. 145-149.

[140] Sakai, H., "A Method for Entity-Relationship Behaviour Modelling", in *Entity-Relationship Approach to Software Engineering* (the Third International Conference on E-R Approach, California, 1983), edited by C. G. Davis, S. Jajodia, P. A. Ng and R. T. Yeh, Elsevier Science Publishers B. V. (North-Holland), 1983a, pp. 111-129.

[141] Sakai, H., "E-R Approach to Logical Database Design", in *Entity-Relationship Approach to Software Engineering* (the Third International Conference on E-R Approach, California, 1983), edited by C. G. Davis, S. Jajodia, P. A. Ng and R. T. Yeh, Elsevier Science Publishers B. V., North Holland, 1983b, pp. 155-187.

[142] Scheuermann, P., Schiffner, G., and Weber, H., "Abstraction Capabilities and Invariant Properties Modelling within the Entity-Relationship Approach", in *Entity-Relationship Approach to System Analysis and Design* (the First International Conference on E-R Approach), edited by P. P. Chen, North-Holland Publishing Co., 1980, pp. 121-140.

[143] Schrefl, M., Tjoa, A. M., and Wagner, R. R., "Comparison-Criteria for Semantic Data Models", *International Conference on Data Engineering*, 1984, pp. 120-124.

[144] Segev, A., "Transitive Dependencies", in the Surveyors' Forum of *ACM Computing Surveys*, Vol. 19, No. 2, June 1987, pp. 191-193.

[145] Shepherd, A., and Kerschberg, L., "Constraint Management in Expert Database Systems", in *Expert Database Systems* (Proc. form the First International Workshop), edited by Larry Kerschberg, Benjamin/Cummings Publishing Co., 1986, pp. 309-331.

[146] Simon, E., and Valduriez, P., "Design and Implementation of an Extendible Integrity Subsystem", *ACM-SIGMOD Proc. International Conference on Management of Data*, Boston, Massachusetts, June 1984, pp. 9-17.

[147] Smith, J. M., and Smith, D. C. P., "Database Abstractions: Aggregation and Generalization", *ACM Transactions on Database Systems*, Vol. 2, No. 2, June 1977a, pp. 105-133.

[148] Smith, J. M., and Smith, D. C. P., "Database Abstractions: Aggregation", *Communications of the ACM*, Vol. 20, No. 6, June 1977b, pp. 405-413.

[149] Solvberg, A., and Kung, C. H., "On Structural and Behavioral Modelling of Reality", in *Database Semantics (DS-1)*, edited by T.B. Steel, Jr. and R. Meersman, Elsevier Science Publishers, B.V. (North-Holland), 1986, pp. 205-221.

[150] Spivey, J. M., *Understanding Z*, Cambridge University Press, 1988.

[151] Stonebraker, M., "Implementation of Integrity Constraints and Views by Query Modification", *Proc. of ACM SIGMOD International Management of Data*, San Jose, May 1975, pp. 65-78.

[152] Storey, V. C., *View Creation: An Expert View Creation System for Database Design*, Ph.D. Dissertation, Faculty of Commerce and Business Administration, University of British Columbia, Vancouver, B.C., Canada, October 1986, ICIT Press,1988.

[153] Storey, V. C., and Goldstein, R. C., "A Methodology for Creating User Views in Database Design", *ACM Transactions on Database Systems*, Vol. 13, No. 3, September 1988, pp. 305-338.

[154] Storey, V. C. and Goldstein, R. C., "Design and Development of an Expert Database Design System", *International Journal of Expert Systems: Research and Applications*, Vol.3, No.1, 1990, pp. 31-63.

[155] Storey, V. C., and Goldstein R. C., "Knowledge-Based Approaches to Database Design", Working Paper, University of Rochester, 1991.

[156] Studer, R., "A Conceptual Model for Physical and Logical time", in *Entity-Relationship Approach* (the Sixth International Conference on E-R Approach, New York , Nov. 1987), edited by S. T. March, Elsevier Science Publishers, B.V. (North-Holland), 1988, pp. 223-235.

[157] Su, S. Y. W., and Raschid, L. "Incorporating Knowledge Rules in a Semantic Data Model: An Approach to Integrated Knowledge Management", *the Second Conference on Artificial Intelligence Application*, Miami, Florida, December 1985, pp. 250-256.

[158] Tauzovich, B., "An Expert System for Conceptual Data Modelling", in *the Eighth International Conference on E-R Approach*, Toronto, Canada, October 1989.

[159] Tauzovich, B., "Towards Temporal Extensions to the Entity-Relationship Model", in *the Tenth International Conference on E-R Approach*, San Mateo, California, October, 1991, (proceedings edited by T. J. Teorey), pp. 163-179.

[160] Teorey, T. J., Yang, D., and Fry, J. P., " A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model", *Computing Surveys*, Vol. 18, No. 2, June 1986, pp. 197-222.

[161] Thompson, J. P., *Data with Semantics: Data Models and Data Management*, Van Nostrand Reinhold, N.Y., 1989.

[162] Troyer, O. D., "RIDL*: A Tool for the Computer-Assisted Engineering of Large Databases in the Presence of Integrity Constraints", *Proc. of ACM SIGMOD International Management of Data*, Oregon, June 1989, pp. 418-429.

[163] Tsichritzis, D. C., and Lochovsky, F. H., *Data Models*, Prentice-Hall Inc., 1982.

[164] Ullman, J. D., *Principles of Database Systems*, the second edition, Computer Science Press, Rockville, Maryland, 1982.

[165] Urban, S. D., and Delcambre, L. M. L., "An Analysis of the Structural Dynamic, and Temporal Aspects of Semantic Data Models", *The Second International Conference on Data Engineering*, Los Angeles, California, February, 1986, pp. 382-389.

[166] Urban, S. D., and Delcambre, L. M. L., "Constraint Analysis for Specifying Perspectives of Class Objects", *The Fifth International Conference on Data Engineering*, Los Angeles, California, February 1989, pp. 10-17.

[167] van der Lans, R. F., *The SQL Standard — A Complete Reference*, originally in Dutch, translated into English by Andrea Gray, original Dutch published by Academic Science, Schoonhoven, 1988 (translated English version published by Prentice Hall, 1989).

[168] Wagner, C., *View Integration in Database Design* Unpublished Ph.D. Dissertation, The University of British Columbia, Vancouver, B.C., Canada, April 1989.

[169] Wand, Y., and Weber, R., "An Ontological Analysis of Some Fundamental Information Systems Concepts", *Proc. of the Ninth International Conference on Information Systems*, Minneapolis, Mn., 1988, pp. 213-225.

[170] Wand, Y., and Weber, R., "An Ontological Analysis of Some Systems Analysis and Design Methods", in *Information Systems Concepts — An In-Depth Analysis*, edited by E. Falkenberg and P. Lindgreen, North-Holland Publishing Co., Amsterdam, 1989, pp. 79-107.

[171] Wand, Y., and Weber, R., "Toward a Theory of the Deep Structure of Information Systems", *Proc. of the Eleventh International Conference on Information Systems*, Copenhagen, December 1990.

[172] Weber, W., Stucky, W., and Karszt, J., "Integrity Checking in Data Base Systems", *Information Systems*, Vol. 8, No. 2, 1983, pp. 125-136.

[173] Webre, N.W., "An Extended Entity-Relationship Model and its Use on a Defense Project", in *Entity-Relationship Approach to Information Modelling and Analysis* (the second International Conference on E-R Approach, 1981), edited by P. P. Chen, Elsevier Science, (North-Holland Publishing Co.), 1983, pp. 173-193.

[174] Wilmot, R. B., "Foreign Keys Decrease Adaptability of Database Designs", *Communications of the ACM*, Vol. 27, No. 12, December 1984, pp. 1237-1243.

[175] Wong, H. K.T., and Mylopoulos, J., "Two Views of Data Semantics: A Survey of Data Models in Artificial Intelligence and Database Management", *INFOR*, Vol. 15, No. 3, October 1977, pp. 344-383.

[176] Yang, H.-L., and Goldstein, R. C., "Identification of Semantic Integrity Constraints for Database Design", Working Paper, 89-MIS-021, Faculty of Commerce and Business Administration, The University of British Columbia, 1989.

[177] Yang, H.-L., "Semantic Integrity Constraint Representation Model: Some Illustrative Examples", Working Paper, Faculty of Commerce and Business Administration, The University of British Columbia, February 1992.

[178] Yasdi, R., and Ziarko, W., "Conceptual Schema Design: A Machine Learning Approach", in *Methodologies for Intelligent Systems*, edited by Z. W. Ras and M. Zemankova, Elsevier Science Publishers Co., 1987, pp. 379-391.

[179] Zvieli, A., and Chen, P. P., " Entity-Relationship Modelling and Fuzzy Databases", *The Second International Conference on Data Engineering*, Los Angeles, California, February 1986, pp. 320-327.

Appendix A

BNF Descriptions of the SIC Representation Model

This appendix provides a summary of the syntax of the SIC Representation model used in this research. The following usual BNF meta-symbols are used: < > ::= { } [ ] | If a terminal symbol happens to be identical to a meta-symbol, it will be written within a pair of escape symbols, e.g., ‡|‡ representing the terminal |.

<SIC statement> ::= [*<SIC name>*]
CERTAINTY *<certainty factor>*
FOR *<object type name>*
ON *<operation type>*
[IF *<assertions>*]
ASSERT *<assertions>*
ELSE *<violation action>*

<SIC name> ::= *<object type name>*-*<operation abbreviation>*-
<SIC type>[-< *optional part* >]

SIC type is specified by the automated database design system or the database designer using some conventions.

<optional part> ::= *<related object type set>*[-*<sequence number>*]

<object type name> ::= *<entity type name>*
| *<relationship type name>*
| *<relation type name>*
| *<attribute type name>*

<attribute type name> ::= *<entity type name>*.*<attribute name>*
| *<relationship type name>*.*<attribute name>*
| *<relation type name>*.*<attribute name>*

Entity type name, relationship type name, relation type name, and *attribute name* are all specified by the database designer, and by convention, begin with a capital letter.

<operation abbreviation> ::= I | D | U

<*related object type set*> ::= (<*related object*> {, <*related object*>})

<*related object*> ::= <*object type name*> | <*Prolog variable*>

Prolog variable is any name beginning with a capital letter or an underline sign (following the Prolog convention). In this case, it is used to represent a single object type name or a set including some object type names.

<*sequence number*> ::= 1 | 2 | 3 | ...

Note that they are positive integer numbers.

<*certainty factor*> ::= <*binary certainty factor*>
| <*fuzzy certainty factor*>
| <*certainty number*>

<*binary certainty factor*> ::= certain | uncertain

Note that "certain" is equivalent to the ratio certainty number 100%, "uncertain" expresses an unknown ratio certainty number that is less than 100%.

<*fuzzy certainty factor*> ::= usually | sometimes

Note that the database designer would specify these *fuzzy certainty factors* to be equivalent to some certainty numbers (for example, "usually" may be set to 80%).

<*certainty number*> ::= <*ordinal certainty number*>
| <*ratio certainty number*>

<*ordinal certainty number*> ::= 1 | 2 | 3 | ...

Note that they are positive integer numbers.

<*ratio certainty number*> ::= 1% | ... | 100%

Note that they are positive real number from 1% to 100% inclusive and are represented in percentage.

<*operation type*> ::= insertion | deletion | update

<*assertions*> ::= [<*quantifier*> <*variable*> <*and_connective*>]
<*assertion_and_list*> [<*or_connective*> <*assertions*>]

<*assertion_and_list*> ::= <*expression*> [<*and_connective*> <*assertion_and_list*>]

<*expression*> ::= (<*assertions*>)
| <*not*> (<*assertions*>)
| <*logical predicate*>
| <*set expression*>
| <*arithmetic expression*>
| <*date expression*>

<*logical predicate*> ::= <*predicate name*>[(<*argument*> {,<*argument*>})]

Predicate name is any name beginning with a lower-case letter following the usual Prolog convention. The built-in logical predicates for storing or manipulating information in data dictionary are listed in Appendix B.

<*not*> ::= not | ¬

<*and_connective*> ::= ∧ | ,

<*or_connective*> ::= ∨ | ;

<*quantifier*> ::= ∀ | ∃

<*argument*> ::= <*constant*> | <*EntRshipRel variable value*>
| <*arithmetic simple expression*>
| <*date simple expression*>

<*constant*> ::= <*numeric constant*>
| <*string*>
| <*atom*>
| <*date constant*>

Atom is the same as in Prolog, i.e. it is made up of letters and digits, and begins with a lower-case letter; *string* is a character string.

<*numeric constant*> ::= <*integer number*> | <*real number*>

<*date constant*> ::= <*time point*> | <*time interval*>

Time point is a specific date following any date representation convention within a pair of double quotation marks ("") (the system takes responsibility for deciphering the input string and interpreting it as a date); *time interval* is a period represented by a real number following one time unit, i.e., second(s), minute(s), hour(s), day(s), month(s), and year(s) within a pair of quotation marks (""); e.g., "3.5 hours".

<*variable*> ::= <*EntRshipRel variable*> | <*other_variable_except_ERR*>

<*EntRshipRel variable*> ::= <*entity type name*>$_{[<subscript>]}$
| <*relationship type name*>$_{[<subscript>]}$
| <*relation type name*>$_{[<subscript>]}$

<*subscript*> ::= 0 | 1 | 2 | ...

Note that they are non-negative integer numbers.

<*other_variable_except_ERR*> ::= <*attribute variable*>
| <*Prolog variable*>

<*attribute variable*> ::= <*entity type name*>$_{[<subscript>]}$.<*attribute name*>
| <*relationship type name*>$_{[<subscript>]}$.<*attribute name*>
| <*relation type name*>$_{[<subscript>]}$.<*attribute name*>

<*set expression*> ::= [set] ‡{‡<*variable*> ‡|‡<expression>‡}‡

<*arithmetic simple expression*> ::= [<*positive_negative_sign*>] <*arithmetic term*>
{<*arithmetic_plus_minus*> <*arithmetic term*>}

<*arithmetic term*> ::= <*arithmetic subterm*>
{<*arithmetic_times_divide*> <*arithmetic subterm*>}

<*arithmetic subterm*> ::= <*numeric constant*>
| <*other variable value*>
| <*aggregate function*>(<*other variable value*>)
| count(<*EntRshipRel variable value*>)
| <*aggregate function*>(<*set expression*>)
| (<*arithmetic simple expression*>)
[<*exponential operator*> (<*arithmetic simple expression*>)]

<*other variable value*> ::= <*other_variable_except_ERR*>
| new(<*other_variable_except_ERR*>)
| old(<*other_variable_except_ERR*>)

<*EntRshipRel variable value*> ::= <*EntRshipRel variable*>
| new(<*EntRshipRel variable*>)
| old(<*EntRshipRel variable*>)

<*positive_negative_sign*> ::= + | −

<*arithmetic_plus_minus*> ::= + | −

<*arithmetic_times_divide*> ::= × | /

<*exponential operator*> ::= ↑

<*aggregate function*> ::= sum | avg | min | max | count
| <user-defined aggregate function>

<*arithmetic expression*> ::= <*arithmetic simple expression*>
[<*not*>] <*comparison operator*> <*arithmetic simple expression*>

<*comparison operator*> ::= ≤ | ≥ | < | > | = | ≠

<*date simple expression*> ::= [<*positive_negative_sign*>] <*date subterm*>
{<*date operator*> <*date subterm*>}

<*date subterm*> ::= <*date constant*>
| <*other variable value*>
| <*date function*>(<*other variable value*>)
| (<*date simple expression*>)

<*date operator*> ::= + | −

<*date function*> ::= year | month | day | minute | second

<*date expression*> ::= <*date simple expression*>
[<*not*>] <*comparison operator*> <*date simple expression*>

<*violation action*> ::= <*rejection*> | <*propagation*> | warning

<*rejection*> ::= reject | conditionally_reject

<*propagation*> ::= propagate(<*propagated_action*>)
| conditionally_propagate(<*propagated_action*>)

<*propagated_action*> ::= insert(<*variable*>)
| delete(<*variable*>)
| insert_all(<*variable*>)
| delete_all(<*variable*>)
| insert(<*entity role type*>, <*variable*>)
| delete(<*entity role type*>, <*variable*>)
| insert_all(<*entity role type*>, <*variable*>)
| delete_all(<*entity role type*>, <*variable*>)
| update(<*variable*>, <*arithmetic simple expression*>)
| update(<*variable*>, <*date simple expression*>)

<*entity role type*> ::= "<*entity type name*>"
| <*Prolog variable*>

Prolog variable is used to represent an entity type name in this case.

Appendix B

Summary of the Predicates used in this Research

This appendix provides a summary of the logical predicates used with the SIC Representation model. The listing is not intended to be comprehensive. All the "input predicates" (in Appendix B.1) and most of the "manipulation predicates" (e.g., *ent_occ*, in Appendix B.2) are used only in conjunction with generic SICs. Other manipulation predicates (e.g., *rship_occ_part*, *is_null*) are represented as Prolog "sub-procedures", which are convenient for SIC representation. However, it is possible to represent a specific SIC without using any of these predicates.

B.1 Input Predicates

The following predicates are suggested by this research for conveying semantic information about a database. They are used by a database designer to indicate that there are some integrity constraints on specific objects, which would inherit relevant generic SICs. They are information provided by the database designer and stored in a database.

- The predicate *domain(Domain_Name, Data_Type, Format, Value_Range)* is used to define a user-defined domain, *Domain_Name*, e.g., *money, location*; its system pre-defined *Data_Type*, e.g., *arithmetic, date, non_arithmetic*; its format; and value range. Stating that a data type is of type arithmetic if all of the normal arithmetic operations may be performed on it in the usual way. Specifying a data with a non_arithmetic data type implies that the data may not be used in conventional arithmetic operations. Following the idea of the EBASE system to simplify the database designer's work ([Goldstein and Wagner, 1988]), no format specification is needed for date and arithmetic data type domains. For non_arithmetic data types, the format is comprised of four basic symbols enclosed in the pair of quotation marks:

  A for an alphabetic character, i.e., A to Z, a to z, period or blank;
  9 for a numeric character, i.e., 0 to 9;
  ! for any special character, e.g., - / ,. [] { } ! etc. or blank;
  X for an alpha-numeric character (i.e., any of the above).

  If a format contains a string of one type of character, it can be expressed more

concisely as "7@X", which means "XXXXXXX". The range is expressed as a continuous range from *Beginning_value* to *End_value* in the format of [*Beginning_value .. End_value*], or an enumerated set. Following the usual mathematic convention, the symbols are used here as follows: "[" means the beginning of a closed range, i.e., including the *Beginning_value*; "]" means the end of a closed range, i.e., including the *End_value*; "(" means the beginning of an open range, i.e., excluding the *Beginning_value*; ")" means the end of an open range, i.e., excluding the *End_value*. The special symbol "*" is used if there is no upper or lower bound.

- The predicate *attribute(Entity/Relationship_Type, Attribute_Name, Domain_Name, Special_Value_Range, Null?, Unique?, Candidate_Key_Attribute?, Changeable?)* is used to specify information about an attribute of an entity or relationship type. *Domain_Name* is its value domain definition. *Special_Value_Range* is its specific value range information (e.g., *Employee.Salary* has the domain of money with the specific value range between $2,000 and $100,000). The four binary variables, *Null?, Unique?, Candidate_Key_Attribute?, Changeable?* indicate whether the attribute is allowed to be null, unique, a part of a candidate key, and changeable.

- The predicate *entity(Entity_Type, Primary_Key, Composite_Key_Set, Absolute_Max_Cardinality)* is used to describe *Entity_Type*. *Primary_Key* specifies its primary key. *Composite_Key_Set* is a set comprised of all composite keys (including primary and non-primary composite keys). *Absolute_Max_Cardinality* specifies the maximum number of occurrences that are allowed in the *Entity_Type*.

- The predicate *relationship_participant(Relationship_Type, Entity_Type, Min_Cardinality, Max_Cardinality)* specifies *Relationship_Type*'s participant, *Entity_Type*, and the usual relationship cardinalities relative to it. This predicate is used for each relationship in the E-R-SIC model. In the relational model, it is used for those relationships that are separately represented. If a relationship is hidden in an entity relation, its two related *relationship_participant* predicates are deleted, and another *relationship_hidden_entity* predicate is created (see below).

- The predicate *relationship_hidden_entity(RelationshipType, EntityType1, ForeignEntType, MinCard2, MaxCard2)* is used in the relational model to indicate that *RelationshipType* is represented via a foreign key in the *EntityType1* relation. It also indicates that relative minimum and maximum cardinalities of *ForeignEntType* in the *RelationshipType* are *MinCard2* and *MaxCard2*, respectively.

- The predicate *symmetric(Relationship)* is used to indicate that a relationship is symmetric.

- The predicate *transitive(Relationship)* is used to indicate that a relationship is transitive.

- The predicate *subset_rship((RshipType1,EntType1,EntType2), (RshipType2, EntType3, EntType4))* is used to indicate that the first relationship type is a subset of the second relationship type in the sense that *EntType3* and *EntType4* are the participants in the second relationship type corresponding respectively to *EntType1* and *EntType2* in the first relationship type. Usually, *EntType1* is the same as *EntType3*, and *EntType2* is the same as *EntType4*. However, it is possible that *EntType1* and *EntType3* are different subtypes of a super-type, and *EntType2* and *EntType4* are different subtypes of another super-type (these two super-types may be the same). In that case, if an occurrence *E1* of *EntType1* connects with an occurrence *E2* of *EntType2* in *RshipType1*, an *EntType3* occurrence, corresponding to the super-type occurrence of *E1*, should also connect with an *EntType4* occurrence, corresponding to the super-type occurrence of *E2* in *RshipType2*.

- The predicate *rships_union_condition((SuperRship,EntType1, EntType2), SubRshipSet)* is used to indicate a necessary condition for the case that the first relationship type is the union of the other relationship types that are in the set, i.e., the second argument of the predicate is a set in the form of {*(SubRshipi, EntTypei, EntTypej)*}, *i,j= 1,2, ..., i≠ j*. Similarly to the case of the predicate *subset_rship*, usually *EntTypei*'s are the same as *EntType1*, and *EntTypej*'s are the same as *EntType2*. It is also possible that *EntTypei*'s and *EntType1* are different subtypes of a super-type, and *EntTypej*'s and *EntType2* are different subtypes of another super-type (these two super-types may be the same). In that case, if there is an *SuperRship* occurrence connecting an occurrence *E1* of *EntType1* with an occurrence *E2* of *EntType2*, there should be at least one *SubRshipi* occurrence connecting an *EntTypei* occurrence, corresponding to the super-type occurrence of *E1*, with an *EntTypej* occurrence, corresponding to the super-type occurrence of *E2*.

- The predicate *rships_intersect_condition((SubRship, EntType1, EntType2), SuperRshipSet)* is used to indicate that the first relationship type is the intersection of the other relationship types that are in the set. The second argument of the predicate, *SuperRshipSet*, is a set in the form of {*(SuperRshipi, EntTypei, EntTypej)*}, *i,j= 1,2, ..., i≠ j*. Similarly to the case of the predicate *rships_union_condition*, this kind of SIC may occur in a specialization hierarchy. In that case, for each of *SuperRshipi* to have an occurrence connecting an *EntTypei* occurrence, corresponding to a common super-type occurrence (say, $\widehat{Ek}$), with its *EntTypej* occurrence, also corresponding to a common super-type occurrence (say, $\widehat{Fk}$), there should be an *SubRship* occurrence connecting an *EntType1* occurrence, corresponding to $\widehat{Ek}$, with an *EntType2* occurrence, corresponding to $\widehat{Fk}$.

- The predicate *ex_rships(ExRshipSet)*, where *ExRshipSet* has the form of {*(Rship1, SharingEnt1), (Rship2, SharingEnt2)*}. The *SharingEnt1* and *SharingEnt2* are usually the same. In this case, the predicate is used to indicate that an occurrence of

the sharing entity type is allowed to participate in only one of these two relationship types. The *SharingEnt1* and *SharingEnt2* may be different subtypes of a super-type in a specialization hierarchy. In that case, this predicate is used to indicate that if an occurrence E of a sharing entity type participates in one of these two relationship types, the occurrence of the other sharing entity type, corresponding to the super-type occurrence of E, cannot participate in the other relationship type.

- The predicate *ex_occ(ExoccRshipSet)*, where *ExoccRshipSet* has the form of {*(RshipType1, EntType1, EntType2), (RshipType2, EntType3, EntType4)*}, is used to indicate that there should not be any common occurrences of these two relationship types in the sense that *EntType3* and *EntType4* are the participants in the second relationship corresponding respectively to the *EntType1* and *EntType2* in the first relationship. Similarly to the case of the predicate *ex_rships(ExRshipSet)*, *EntType1* and *EntType3* may be different subtypes of a super-type and *EntType2* and *EntType4* may also be different subtypes of another super-type.

- The predicate *not_and_rships(NotAndRshipSet)*, where *NotAndRshipSet* has the form of {*(Rshipi, SharingEnti)*}, *i=1,2, ...* Similarly to the case of the predicate *ex_rships(ExRshipSet)*, the *SharingEnti*'s are usually the same for *Rshipi*'s. In this case, the predicate is used to indicate that an occurrence of the sharing entity type cannot participate in all of these relationship types. The *SharingEnti*'s may also be different subtypes of a super-type in a specialization hierarchy. In that case, at least one *SharingEnti* occurrence, corresponding to the same super-type occurrence, cannot participate in its *Rshipi*.

- The predicate *either_rships(EitherRshipSet)*, where *EitherRshipSet* has the form of {*(Rshipi, SharingEnti)*}, *i=1,2, ...* Similarly to the case of the predicate *ex_rships(ExRshipSet)*, the *SharingEnti*'s are usually the same for *Rshipi*'s. In this case, the predicate is used to indicate that an occurrence of the sharing entity type must participate in at least one of these relationship types. If the *SharingEnti*'s are different, at least one occurrence of *SharingEnti*, corresponding to the same super-type occurrence, must participate in its *Rshipi*. It is desirable to have two sub-types of these SICs with different violation actions; one for the case with only two relationships, and one for the case with more than two relationships. However, the predicate can be the same.

- The predicate *before((Rship1, SharingEnt1), (Rship2, SharingEnt2))* is used to indicate that at the time an occurrence of *SharingEnt1* is going to participate in *Rship1*, the occurrence of *SharingEnt2*, corresponding to the same super-type occurrence, must participate in *Rship2*.

- The predicate *not_before((Rship1, SharingEnt1), (Rship2, SharingEnt2))* is used to indicate that at the time an occurrence of *SharingEnt1* is going to participate in *Rship1* type, the occurrence of *SharingEnt2*, corresponding to the same super-type occurrence, must not participate in *Rship2*.

- The predicate *rships_join(RshipType1, EntType1, EntTypeN), RshipList)* is used to indicate that if there is a linking path via those relationship types in *RshipList* to connect two entity occurrences in *EntType1* and *EntTypeN* together, these two entity occurrences must be connected via *RshipType1*. *RshipList* is an ordered set (list) in the form of *(RshipTypei, EntTypej, EntTypek)* where *i, k= 2, 3, ..., N* and *j= 1, 2, ..., N_1*. It is a necessary condition for asserting that *RshipType1* is the join of these *RshipTypei*'s, that is, *RshipType1*[*EntType1,EntTypeN*]= *RshipType2*[*EntType1,EntType2*] ⋈ *RshipType3*[*EntType2,EntType3*] ⋈ ... ⋈ *RshipTypeN*[*EntTypeN_1,EntTypeN*]. Since its representation is complicated, the unusual cases in a specialization hierarchy are not considered.

- The predicate *rship_dep_loopn_rships((RshipType1, EntType1, EntTypeN), RshipList)* is used to indicate that if an occurrence of *RshipType1* exists, there should be a linking path via other relationship types in *RshipList* to connect its participating entity occurrences (in *EntType1* and *EntTypeN*) together. *RshipList* is an ordered set (list) in the form of *(RshipTypei, EntTypej, EntTypek)* where *i, k= 2, 3, ..., N* and *j= 1, 2, ..., N_1*. Both this and the above *rships_join* predicates are needed to guarantee that *RshipType1* is the join of these *RshipTypei*'s. That is, *RshipType1*[*EntType1,EntTypeN*]= *RshipType2*[*EntType1,EntType2*] ⋈ *RshipType3*[*EntType2,EntType3*] ⋈ ... ⋈ *RshipTypeN*[*EntTypeN_1,EntTypeN*]. Since its representation is complicated, the unusual cases in a specialization hierarchy are not considered.

- The predicate *weak_entity(EntityType, RelationshipType)* is used to indicate that the entity type is weak and *RelationshipType* is the one that it depends on.

- The predicate *id_depend(EntityType, KeyAttSet, RelationshipType)* is used to indicate that because of the semantics of *RelationshipType* (e.g., implying ID dependency or as an *is_a* relationship, etc.), *EntityType* incorporates the primary key of the other participating entity type as (part of) its candidate key attributes *KeyAttSet*.

- The predicate *critical_rship(RelationshipType, EntType, CriticalAtt)* is used to indicate that not only *RelationshipType* is total to *EntType*, but also exactly one critical relationship occurrence exists for each occurrence of *EntType*. *CriticalAtt* is a binary attribute in *RelationshipType* to indicate whether a relationship occurrence is critical.

- The predicate *completeness_mapping(RelationshipType)* is used to indicate that a relationship type is complete, which means that for each occurrence of one entity type, all occurrences of the other entity type must relate to it via this relationship type.

- The predicate *rship_trigger_rship((Rship1, SharingEnt1), (Rship2, SharingEnt2))* is used to indicate that if an occurrence of *Rship1*, in which an occurrence *E* of

SharingEnt1 participated, existed in the past (and no longer exist now), the *SharingEnt2* occurrence corresponding to the super-type occurrence of E must participate in *Rship2*.

- The predicate *ex_ents(ExEntSet)* is used to indicate that the two entity types in the set are exclusive. The *ExEntSet* contains only two entity types. If three or more entity types are mutually exclusive, we can specify more than one *ex_ents* predicate.

- The predicate *ents_intersect_condition(SubEntType, SuperEntSet)* is used to indicate a necessary condition for the case that *SubEntType* is the intersection of the entity types in the set, *SuperEntSet*. That is, for each of the entity types in *SuperEntSet* to have an occurrence with the same candidate key value, there should be a corresponding occurrence with this candidate key value in *SubEntType*.

- The predicate *ents_union_condition(SuperEntType, SubEntSet)* is used to indicate a necessary condition for the case that the *SuperEntType* is the union of the entity types in the set, *SubEntSet*. That is, for any occurrence in *SuperEntType*, there should be at least one corresponding occurrence with the same candidate key value in one of those entity types that are in *SubEntSet*.

B.2 Manipulation Predicates

The following predicates are used to manipulate the information contained in the input predicates. In order to represent all generic SICs mentioned in this dissertation, an automated database design system should have at least these built-in predicates. In the following, only the functioning of these predicates is described. The actual Prolog code is not provided except for those complicated recursive predicates — *join_list_ok1* and *join_list_ok2*. All predicates are applicable to the E-R representation and the relational representation unless explicitly stated otherwise. The Prolog representation of an occurrence would include its associated entity or relationship type and its primary key (e.g., *("Person", "123-456-789")*).

- The predicate *"rship_occ_part(R,RoleType,E)"* is used to evaluate whether an entity occurrence E participates in a relationship occurrence R with the *RoleType*. Note that usually the entity type of E is the same as the *RoleType*. However, in a specialization hierarchy the entity type of E may be *E_Type* and the *RoleType* may be *F_Type*, where *E_Type* and *F_Type* have a common super-type. In that case, the predicate means that the F occurrence in *F_Type*, which corresponds to the E occurrence in the *E_Type*, participates in the relationship occurrence R.
  For example, if a SIC originally obtained from the database designer refers to a

relationship such as "*E R F*" and *E* is the only sharing entity type, it will be written as *rship_occ_part(R, "E",E)* in its sub-SICs.
The exact functioning of this predicate *"rship_occ_part(R,RoleType,E)"* is as follows. *RoleType* must be instantiated.
(1)Suppose that both *R* and *E* are instantiated. (1a) Suppose *E_Type* is the same as *RoleType*. This predicate checks whether the primary key value of the occurrence *E* and the corresponding key attributes of the occurrence *R* are equivalent. It returns false if they are not equivalent or if the corresponding key attributes of the occurrence *R* are null. (1b) Suppose *EType* is different from *RoleType*. It traverses a specialization hierarchy to find the corresponding occurrence in *RoleType* and check that occurrence with *R*.
(2) Suppose that *E* is uninstantiated, *R* is instantiated. It returns the occurrence participating in *R* with *RoleType* as *E*. No traverse of a specialization hierarchy would be performed.
(3) Suppose that *E* is instantiated, *R* is not. It returns the *R* (*R*s when backtracking) in which *E* participates with the *RoleType*.

- The predicate *ent_occ(Entity_Type, E)* is used to evaluate whether an *E* is an occurrence of *Entity_Type*, or used to fetch any one occurrence *E* from *Entity_Type*. If *E* is null, the logical value of this predicate is undefined. It does not traverse a specialization hierarchy to find the corresponding entity occurrence. For example, *ent_occ("Manager", Engineer)* would be false though the mentioned engineer may also appear in the manager entity type.

- The predicate *rship_occ(Relationship_Type, R)* is used to evaluate whether *R* is an occurrence of *Relationship_Type*, or used to fetch any one occurrence from *Relationship_Type*. If *R* is null, the logical value of this predicate is undefined.

- The predicate *att_occ(E, Att_Name, E.A)* (or *att_occ(R, Att_Name, R.A)*) is used to get the value of an attribute occurrence *E.A* (or *R.A*) of an entity occurrence *E* (or relationship occurrence *R*). If *Att_Name* is a key attribute of a relationship type, *att_occ* can be used to reference its value.

- The predicate *comp_atts_occ(E, Comp_Att, E.C)* is used to get the value of a composite attribute occurrence, that is, by applying a "concatenate" operator to the attribute values in the ordered set, *Comp_Att*.

- The predicates *satisfy_datatype(E.A, Data_Type), satisfy_format(E.A, Format), satisfy_value(E.A, Range),* are used to determine whether *E.A* satisfies the corresponding data_type, format, value, respectively.

- The *is_null(x)* predicate is evaluated to be true if and only if *x* is "null".

- The *is_not_null(x)* predicate is evaluated to be true if an only if *x* is not "null".

- The *false* predicate is used to indicate that the assertion is absolutely false. That is, it is used to indicate that the attempted data operation — the operation T component in the SIC Representation model — is not allowed.
- The predicate *concatenate(String1, String2, Resulting_String)* is used to construct *Resulting_String* by appending *String2* after *String1*.
- The predicate *substring(String, Beginning_Position, End_Position, Resulting_Substring)* is used to get *Resulting_Substring* from *Beginning_Position* to *End_Position* of *String*.
- The predicate *concatenate_SICname(String1, Variable, String2, Resulting_SICname)* is used to construct *Resulting_SICname* by appending the value of *Variable* after *String1*, and then concatenating with *String2*.
- The predicate *checkcomSIC(Component_SIC_Name, Checked_Occurrence)* calls a SIC named *Component_SIC_Name* to check whether *Checked_Occurrence* satisfies it; if not, the violation action of the calling SIC (rather than *Component_SIC_Name*) will be taken. (The SIC aggregation concept is applied.)
- The predicate *checkmemSIC(Member_SIC_Name, Checked_Occurrence)* calls a SIC named *Member_SIC_Name* to check whether *Checked_Occurrence* satisfies it. The violation action of the *Member_SIC_Name* will be taken if the *Checked_Occurrence* violates it. (The SIC association concept is applied.)
- The predicate *not_empty(Set)* is used to test whether *Set* is empty.
- The predicate *belongs_to(Element, Set)* is used to test whether *Element* is in *Set*.
- The predicate *is_compatt(Att, Comp_Key_Set)* is used to test whether an attribute belongs to a composite key. It is different from *belongs_to* because *Comp_Key_Set* may be a coset — its elements are also sets containing some attributes.
- The predicate *part_of_comp_key(Att, Comp_Key_Set, Comp_Key1)* is used to get a composite key *Comp_Key1*, which consists of the attribute *Att*, from *Comp_Key_Set*.
- The predicate *remove_from_set(Element, SourceSet, ResultingSet)* is used to indicate that *ResultingSet* is the result of removing *Element* from *SourceSet*.
- Some special predicates to process a list are needed in handling a **Relationships_Join SIC** or **Relationship_depends_on_LoopN_Relationships SIC**. In the following, *RshipList* is an ordered set (list) in the form of *(RshipTypei, EntTypej, EntTypek)* where *i, j, k= 1, 2, ..., N*.
  - The predicate *precedes(RshipTypej, RshipTypei, RshipList)* is used to find the relationship type, *RshipTypei*, which precedes the relationship type, *RshipTypej*, in *RshipList*.

- The predicate *follows(RshipTypej, RshipTypek, RshipList)* is used to find *RshipTypek*, which follows the relationship type, *RshipTypej*, in *RshipList.*
- The recursive predicate *join_list_ok1(FirstEntType, LastEntType, FirstEntOcc, LastEntOcc, RshipList)* is used to test whether the relationship occurrence connecting the *FirstEntOcc* of the *FirstEntType* with the *LastEntOcc* of the *LastEntType* is equal to the join of a series of entity occurrences participating in *RshipList.*
- The recursive predicate *join_list_ok2(RshipTypej, EntTypej1, EntTypej2, FirstEntOcc, EntOccj1, EntOccj2, LastEntOcc, RshipList)* is another recursive predicate. One relationship occurrence of *RshipTypej, j=1,2,...* in *RshipList*, via which *EntOccj1* of *EntTypej1* connects with *EntOccj2* of *EntTypej2*, is given. This predicate is used to test whether it is possible to produce a relationship occurrence, via which *FirstEntOcc* connects with *LastEntOcc*, and which is equal to the join of corresponding occurrences of these relationship types in *RshipList* with the given relationship occurrence.

Since these two recursive predicate are complicated, their detailed Prolog representations are given as below.

%special recursive predicate in Prolog:
% for $R = R1 \bowtie R2 \bowtie R3 \ldots$
% join_list_ok1 is to test whether
% the given occurrence in R connecting FirstEntOcc with LastEntOcc is equal to
% the join of the corresponding occurrences in R1, R2, ...
%
% join_list_ok2 is to test whether
% for the given occurrence in Rj connecting Entoccj1 with EntOccj2, j=1,2,...
% it is possible to produce a pair of (FirstEntOcc,LastEntOcc)
% that is equal to the join of the corresponding occurrences in R1, R2, ...
%

% join_list_ok1: all variables are inputs
join_list_ok1(FirstEntType, LastEntType, FirstEntOcc,
LastEntOcc, [(RshipTypei, FirstEntType, EntTypei)|RestList]):-
FirstEntType $\neq$ LastEntType,
$\exists$ RshipOcci, rship_occ(RshipTypei, RshipOcci),
rship_occ_part(RshipOcci, FirstEntType, FirstEntOcc),
ent_occ(EntTypei, EntOcci),
rship_occ_part(RshipOcci, EntTypei, EntOcci),
join_list_ok1(EntTypei, LastEntType, EntOcci, LastEntOcc, [RestList]).

```
join_list_ok1(LastEntType, LastEntType, LastEntOcc, LastEntOcc, []).
%

% join_list_ok2: FirstEntOcc and LastEntOcc are output variables,
% others are input variables
join_list_ok2(RshipTypej, EntTypej1, EntTypej2, FirstEntOcc, EntOccj1, EntOccj2,
LastEntOcc, RshipList):-
join_first_half(RshipTypej, EntTypej1, EntOccj1, FirstEntOcc, RshipList),
join_last_half(RshipTypej, EntTypej2, EntOccj2, LastEntOcc, RshipList).

% join_first_half: FirstEntOcc is an output variable, others are input variables
% precedes(+RshipTypej, -RshipTypei, +List) is the predicate to find
% the RshipTypei that precedes RshipTypej in the List.
join_first_half(RshipTypej, EntTypej1, EntOccj1, FirstEntOcc, RshipList):-
precedes(RshipTypej, RshipTypei, RshipList),
∃ RshipOcci, rship_occ(RshipTypei, RshipOcci),
rship_occ_part(RshipOcci, EntTypej1, EntOccj1),
ent_occ(EntTypei, EntOcci),
rship_occ_part(RshipOcci, EntTypei, EntOcci),
join_first_half(RshipTypei, EntTypei, EntOcci, FirstEntOcc, RshipList).

join_first_half(RshipTypej, EntTypej1, EntOccj1, EntOccj1, RshipList):-
¬ precedes(RshipTypej, RshipTypei, RshipList).

% join_last_half: LastEntOcc is an output variable, others are input variables;
% follows(+RshipTypej, -RshipTypek, +List) is the predicate to search for
% the RshipTypek that follows RshipTypej in the List.
join_last_half(RshipTypej, EntTypej2, EntOccj2, LastEntOcc, RshipList):-
follows(RshipTypej, RshipTypek, RshipList),
∃ RshipOcck, rship_occ(RshipTypek, RshipOcck),
rship_occ_part(RshipOcck, EntTypej2, EntOccj2),
ent_occ(EntTypek, EntOcck),
rship_occ_part(RshipOcck, EntTypek, EntOcck),
join_last_half(RshipTypek, EntTypek, EntOcck, LastEntOcc, RshipList).

join_last_half(RshipTypej, EntTypej2, EntOccj2, EntOccj2, RshipList):-
¬ follows(RshipTypej, RshipTypek, RshipList).
```

- The following four predicates are used to store related SICs for updating the primary key of *Relationship_Type* or *Entity_Type*. If a SIC is relevant to the insertion

(or deletion) of *Relationship_Type*, its name is added into the *SIC_Name_Set* in the *associated_PKSICs_I* (or *associated_PKSICs_D*) predicate of the affected key attribute *PKAtt*. Each affected key attribute, *PKAtt*, of *Relationship_Type* has its own *associated_PKSICs_I* (or *associated_PKSICs_D*) predicate. In the case that a SIC is relevant to the insertion (or deletion) of *Entity_Type*, there is only one *associated_PKSICs_I* (or *associated_PKSICs_D*) predicate, which is related to each of its primary key attribute.

  - predicates used to store the related SICs for the key attributes of *Relationship_Type*:
    associated_PKSICs_I(Relationship_Type, PKAtt, SIC_Name_Set)
    associated_PKSICs_D(Relationship_Type, PKAtt, SIC_Name_Set)
  - predicates used to store the related SICs for the key attributes of *Entity_Type*:
    associated_PKSICs_I(Entity_Type, SIC_Name_Set)
    associated_PKSICs_D(Entity_Type, SIC_Name_Set)

- The predicate *foreign_key(RshipType, EntOcc, ForeignEntType, FKAtts_value)* is used in the relational model to get the foreign key value *FKAtts_value* of *EntOcc* connecting with an occurrence of *ForeignEntType* in the relationship *RshipType*.

- The predicate *foreign_ent_occ(RshipType,E,F)* is used in the relational model to evaluate whether the primary key of an F occurrence appears as a foreign key in an occurrence E because of the relationship type, *RshipType*. The *RshipType* is needed because there may be more than one relationship type between two entity types. The entity types with which E and F participate in *RshipType* need not be explicitly included in this predicate. From the relationship name *RshipType*, the reference to *relationship_hidden_entity(RshipType, EntType1, EntType2, Cmin, Cmax)* would show that E takes the role *EntType1* and F takes the role of *EntType2*. The exact functioning of this predicate "*foreign_ent_occ(RshipType,E,F)*" is as follows.
  RshipType must be instantiated.
  Suppose that we have *relationship_hidden_entity(RshipType, EntType1, EntType2, Cmin, Cmax)*.
  (1)Suppose that both E and F are instantiated. (1a) Suppose the entity type of E, *EType*, is the same as *EntType1* and the entity type of F, *FType*, is the same as *EntType2*. This predicate checks whether the foreign key value of the occurrence E and the corresponding key attributes of the occurrence F are equivalent. It returns false if they are not equivalent or if the corresponding key attribute of the occurrence E is null. (1b) Suppose either *EType* is different from *EntType1* or *FType* is different from *EntType2*, or both are different. It traverses a specialization hierarchy to find the occurrence in *EntType1* corresponding to E and the occurrence in *EntType2* corresponding to F, and then checks them.
  (2) Suppose that F is uninstantiated, E is instantiated. It only returns F with the

value of the occurrence in *EntType2* whose primary key appears in *E* as a foreign key owing to *RshipType*. No traverse on specialization hierarchy would be performed. (3) Suppose that *F* is instantiated, *E* is not. It returns those *E* (*E*s when backtracking) taking the role of *EntType1* connecting with *F* taking the role of *EntType2*.

- The predicate *which_foreign(EntType, RoleType1, RoleType2, EntOcc1, EntOcc2, EntOcci, FEntOcci)* is used to check which of entity types, *RoleType1* or *RoleType2*, should be *EntType* where the relationship type is hidden, and assign these entity occurrence variables properly. This predicate is used to reduce the possible predefined sub-SICs in the relational model since we handle the entity links in specialization hierarchy implicitly and for some SICs (e.g., **Subset_Relationship SIC, Exclusive_Occurrence SIC**) the information about the corresponding involved pairs of sharing entity types are stored in some order.
  EntOcc1 and *EntOcc2* are the occurrences playing *RoleType1* and *RoleType2*, respectively. *EntType* is the entity type where the relationship between *RoleType1* and *RoleType2* is hidden. *EntOcci* is an occurrence of *EntType*, *FEntOcci* is the corresponding foreign entity occurrence. When calling this predicate, *EntType, RoleType1, RoleType2* should be already instantiated. Either the pair of *EntOcc1, EntOcc2* or the pair of *EntOcci, FEntOcci* is already instantiated, but not both. This predicate performs the following:
  (1) if *EntType=RoleType1*, *EntOcc2* should be the foreign entity occurrence, then unify *EntOcci* with *EntOcc1*, and unify *FEntOcci* with *EntOcc2* (that is, (1a) if *EntOcc1, EntOcc2* are already instantiated, then set *EntOcci=EntOcc1, FEntOcci=EntOcc2*, (1b) if instead, *EntOcci, FEntOcci* are already instantiated, then set *EntOcc1=EntOcci, EntOcc2=FEntOcci*);
  (2) conversely if *EntType=RoleType2*, *EntOcc1* should be the foreign entity occurrence then unify *EntOcci* with *EntOcc2*, and unify *FEntOcci* with *EntOcc1*;
  (3) otherwise, return false.

- The predicates *associated_FKSICs_I(Entity_Type, FKAtt, SIC_Name_Set)* and *associated_FKSICs_D(Entity_Type, FKAtt, SIC_Name_Set)* are used in the relational model to store related SICs for updating the foreign key attribute *FKAtt* of *Entity_Type*.

Appendix C

BNF Descriptions of the Simplified Format

This appendix provides a summary of the syntax of the simplified format that is used in this research to represent the preconditions and predicates of a general SIC elicited from the database designer. Since this description is similar to the precondition and predicate components of the SIC Representation model, some definitions refer to Appendix A. The BNF meta-symbols and escape symbols used here are the same as those of Appendix A.

<*SIC description*> ::= if [with_respect_to <*focused entity type names*>,]
<*assertions*> [*previously*]
then [with_respect_to <*focused entity type names*>,]
<*assertions*> [*before*]
| if [with_respect_to <*focused entity type names*>,]
<*assertions*> [*previously*]
then <*SIC description*>
| <*assertions*>

<*focused entity type names*> ::= <*entity type name*>
| (<*entity type name*>, <*entity type name*>)

<*assertions*> ::= [<*quantifier*> <*variable*> <*and_connective*>]
<assertion_and_list> [<*or_connective*> <*assertions*>]

Note that the BNF definitions of *or_connective* and *and_connective* are the same as in Appendix A.

<*quantifier*> ::= ∀
| ∃ [<*numerical modifier*>] [different]

<*numerical modifier*> ::= at_least <*some number*>
| at_most <*some number*>
| exactly <*some number*>

<*some number*> ::= 1 | 2 | 3 | ...

Note that they are non-negative integer numbers.

<*assertion_and_list*> ::= <*expression*> [<*and_connective*> <*assertion_and_list*>]

Note that this BNF definition is the same as in Appendix A except for the underlying definition of *expression*, which is redefined as below.

<*expression*> ::= (<*assertions*>)
| <*not*> (<*assertions*>)
| <*entity type name*> <*relationship type name*> <*entity type name*>
| <*EntRship type name*> is_to_be_deleted
| <*attribute type name*> is_to_be_updated
| <*logical predicate*>
| <*set expression*>
| <*arithmetic expression*>
| <*date expression*>

Entity type name, *relationship type name* and *attribute type name* have the same name convention — beginning with a capital letter — as those in Appendix A. The definitions of *not* and *date expression* are the same as those in Appendix A. The definitions of *set expression* and *arithmetic expression* are almost the same as the corresponding ones in Appendix A except for their underlying definitions of *variable* and *arithmetic subterm*, which will be redefined below. The major differences are that neither subscript nor relation type name is used; and the *new* and *old* functions are only applicable to <*attribute type name*>. Only some of the *logical predicates* in Appendix B are applicable here. These are, *satisfy_datatype(*<*attribute type name*>*, Data_Type), satisfy_format(*<*attribute type name*>*, Format), satisfy_value(*<*attribute type name*>*, Range), is_null(*<*attribute type name*>*),* *is_not_null(*<*attribute type name*>*), false*, and *belongs_to(Element, Set)*, etc. Because subscripts are not used, the simplified format uses a special predicate *unique(*<*attribute type name*>{*,*<*attribute type name*>}*)*, which is not used in the SIC Representation model (so, it is not in Appendix B). It checks whether a particular attribute or combination of attributes does not actually contain duplicate values in a database. When the SIC is represented in the SIC Representation model, this special predicate will be replaced with a *count* function and appropriate subscripts will be attached to its associated entity (by applying the algorithms in Appendix H.2).

<*EntRship type name*> ::= <*entity type name*>
| <*relationship type name*>

<*variable*> ::= <*EntRship type name*> | <*other_variable_except_ER*>

<*other_variable_except_ER*> ::= <*attribute type name*>
| <*Prolog variable*>

<arithmetic subterm> ::= *<numeric constant>*
| *<other variable value>*
| *<aggregate function>*(*<attribute variable value>*)
| count(*<EntRship variable value>*)
| *<aggregate function>*(*<set expression>*)
| (*<arithmetic simple expression>*)
[*<exponential operator>* (*<arithmetic simple expression>*)]

The definitions of *numeric constant*, *aggregate function*, *arithmetic simple expression*, *exponential operator* are all the same as in Appendix A. The main difference of the definition of *arithmetic subterm* is that we use *EntRship variable value* instead of *EntRshipRel variable value.*

<other variable value> ::= *<other_variable_except_ER>*
| new(*<attribute type name>*)
| old(*<attribute type name>*)

<EntRship variable value> ::= *<EntRship type name>*

Note that the *new* and *old* functions are not applicable here. We can directly use *EntRship type name* instead of defining *EntRship variable value.* The reason for defining this is to compare it with Appendix A.

Appendix D

SIC Type Classification in the E-R-SIC Model

This appendix provides a classification of SIC types used in the E-R-SIC model. Although the listing is not comprehensive, it covers different SIC types that can occur in various contexts of an E-R diagram, and also includes those SICs mentioned in the literature. We can use these SIC types to write generic SICs and store some consistency or nonredundancy rules for them in the *elicitation subsystem.*

Notation The following notation will be used in examples.

- *E, F, G, H, I, ...* are used to denote entities.
- *R, RX, RY, RZ, R1, R2, R3, ...* beginning with *"R"* are used to denote relationships.
- *E.A, E.A1, E.A2, F.B, F.B1, F.B2, R.A, ...* are used to denote attributes of entities or relationships. *Current_time* is the system variable to monitor the current clock.
- *v, v1, v2, ...* are used to denote values.
- *comp_op, comp_op1, ...* are used to denote $=, \leq, \geq, \neq, <, >$.
- *ari_op, ari_op1, ...* are used to denote $+, -, \times, \div$.
- *agg_fcn, agg_fcn1, ...* are used to denote the aggregate functions, e.g., *sum, avg, min, max, count.*

SIC Types There are four categories of SIC types.

- **SIC Types Focusing on Attributes of a Single Entity or Relationship Type.**

- **SIC Types Focusing on a Relationship as an Association.** These are necessary conditions for the existence of one or more relationships because relationships are "associations" linking entities. Different contexts, *line, star, loop-2*, and *loop-n* should be considered when identifying SIC types.
- **Sufficient Conditions for the Existence of Relationship(s).** These are sufficient conditions for the existence of one or more relationships because relationships are "associations" linking entities.
- **SIC Types for Entities without Explicit Relationships.** There are some SICs for a group of entities because of some "implicit relationships".

Each of them is described in detail as below.

- **SIC Types Focusing on Attributes of a Single Entity or Relationship Type.**

  1. **Domain Constraint:** declares the value; extended format; nonvolatility[104]; data type of an attribute of an entity or relationship type; whether it can be null; and if it should be unique.

     It is possible to specify this kind of constraint for an "implicit entity or relationship subtype". For example, a **Subtype_Nonvolatility Constraint** requires that an attribute of any occurrence of the specified "implicit entity or relationship subtype " be not changeable. We may also require that an attribute have some value range if there are restrictions on the values of other attribute(s), e.g., **Conditional_Value SIC** such as:
     if E.A1 comp_op1 v1, ... then E.A2 comp_op2 v2.

     Constraints on the attributes of an entity or relationship occurrence could be more complicated. In that case, we would have the following.
  2. **Formula SIC:** requires that a formula[105] exists among attributes of any occurrence of the specified entity or relationship type. For example, we may have:
     (E.A1) comp_op ((E.A2) ari_op1 (E.A3) ari_op2 (scalar_value)),...
     If the formula involves non-numeric data-types, string predicates may be needed.

[104] "Nonvolatility" is interpreted in a strict sense. If an entity is inserted with some null attribute values, they have to stay null if "nonvolatile" is declared for them.

[105] Recall that if there is more than one attribute appearing in an expression, we call that expression as a formula.

There may be some conditions on a formula requiring that it hold only for an "implicit entity or relationship subtype". For example, we may have:
if (E.A1) comp_op1 ((E.A2) ari_op1 (E.A3) ari_op2 (scalar_value)),...
then (E.A4) comp_op2 (E.A5) ari_op3 (E.A3)

3. **Aggregate_Attribute SIC:** restricts the aggregate attributes of the whole entity or relationship set, for example, *agg_fcn(E.A) comp_op v.*
If the data type of the attribute *E.A* is numeric, we may have:
agg_fcn(E.A) comp_op E.A, such as, "*any salary value of an employee must not be more than 5 percent greater than the average of all such values*".

   Some complicated cases may occur. For example, there may be a SIC (called **Aggregate_Attribute_Formula SIC**) requiring some formula among the values of aggregate attributes of the entity or relationship set. There may be a restriction (called **Interdependent_Aggregate_Attributes SIC**) on the aggregate value of an attribute if some other attributes have some aggregate values, e.g.,
   if (agg_fcn1(E.A3) comp_op2 v2), ... then (agg_fcn2(E.A3) comp_op3 v3)

   It is also possible only for occurrences of the specified "implicit entity or relationship subtype" that there is a restriction on an aggregate value (called **Subtype_Aggregate_Attribute SIC**), an aggregate attribute formula (called **Subtype_Aggregate_Attribute_Formula SIC**), or the requirement of the coexistence of some aggregate attributes (called **Subtype_Interdependent_Aggregate_Attributes SIC**).
4. **Composite_Attribute_Unique Constraint:** requires the concatenated values of some attributes (which form a composite key) of an entity type to be unique.
5. **Old_New Transitional Constraint:** restricts the pairs of values before and after an update of an attribute of the specified entity or relationship type. It is also possible to have this kind of constraint for a specified "implicit entity or relationship subtype".
6. **Primary_Key SIC:** requires enforcing a set of SICs related to deletion and insertion of an entity or relationship when an attribute, which is (part of) the primary key, is updated.
7. **Deleted_Object_Attribute SIC:** restricts the values of attributes of any occurrence of the specified entity or relationship type before this entity or relationship occurrence can be deleted. For example, *"a project can be deleted only if its budget is zero"*. It is an operation-dependent SIC that is relevant only to the deletion operation. It is also possible to have this kind of constraint for a specified "implicit entity or relationship subtype".
8. **Absolute Maximum Cardinality Constraint of an Entity Type:** restricts the maximum number of occurrences of an entity type that can exist in

a database. If it is infinite, it is not a restriction. The notation "*" is used to denote either the infinite, or the case that it is not infinite in the mathematical sense, but there is no restriction on the maximum cardinality.

9. **SIC Involving Current_Time:** either restricts a time-valued attribute of the specified entity or relationship type within some range relative to *Current_Time*; places restrictions on other attributes when time-valued attribute(s) satisfy some condition(s) relative to *Current_Time*; or requires the propagation to update the value(s) of some attribute(s) of the specified entity or relationship type while updating the *Current_time.*

   It may only occur in an environment in which the event that causes manipulation of the involved database object(s) is processed in real time so that the *Current_time* in the computer matches the event time in the real world. Otherwise, *Current_time* in all of the above must be replaced by a time-valued attribute that records the external event time, and these constraints become ordinary data-driven constraints.

   - **Attribute SIC with Current_Time Restriction:**
     (a) Restriction on a time-valued attribute relative to *Current_Time*: Four examples are given here to illustrate the various cases.
     * Cases 1 and 2 — the increase of *Current_time* will never violate the SIC under the assumption that the current database is semantically correct:
       i. Case 1 — the time-valued attribute represents a past fact relative to *Current_time*: If "*Employee.FirstWorkDate*" is the first date of *Employee* working on related jobs, *Current_time* $\geq$ *(Employee.FirstWorkDate* + *"2 years")* is an expression to assert that *"each employee must have at least 2 years working experience"*.
       ii. Case 2 — the time-valued attribute represents a future expectancy relative to *Current_time*: If the *"Part.ExpectedArrDate"* is the expected arrival date of a *Part* ordered from *Suppliers*, *Current_time* $\geq$ *(Part.ExpectedArrDate* − *"10 days")* is an expression to assert that *"the expected delivery time of a part must be no more than 10 days"*.
     * Cases 3 and 4 — the increase of *Current_time* may violate the SIC:
       i. Case 3 — the time-valued attribute represents a past fact relative to *Current_time*: We may have an expression *Current_time* $\leq$ *Account_Receivable.Date* + *"10 years"* to assert a rule that *"an account_receivable cannot be older than 10 years"*.
       ii. Case 4— the time-valued attribute represents a future expectancy relative to *Current_time*: If *Drug.Expiration_Date* is the date that a *Drug* will expire, *Current_time* $\leq$ *(Drug.Expiration_Date* − *"2*

weeks") is an expression to assert a rule that *"a drug must be valid for at least 2 weeks"*.

(b) Restriction on other attributes when time-valued attributes satisfy some conditions: When time passes, the restriction will either disappear or come into force. For example,
if Current_time $\leq$ *(Employee.HireDate +* *"6 months")*
then old(Employee.Salary) $\geq$ *new(Employee.Salary)*
is an expression to assert a rule that "*an employee cannot receive a salary raise during his(her) first 6 months in the company*".
Another example is: *"if an employee has worked for at least two years, he (she) must have at least 10 vacation days"*.

- **Current_Time_Triggered SIC on Attributes:** It triggers some operation while updating the *Current_time*. For example, *at 0:00 on 1/1/1993, increase the salary of each employee by $1,000.*

10. **Temporal Conditions Among Attributes:** requires that some attribute(s) of an occurrence of the specified entity or relationship type must satisfy a certain condition, at the time one of its other attribute is going to acquire some value. For example, we may have: *if E.A1 comp-op v1 then E.A2 comp-op v2 before*. Alternatively, this type of SIC may require that if its other attribute(s) satisfied a certain condition in the past (and no longer satisfies it now), one of its attributes must take some value. For example, we may have: *if E.A1 comp-op v1 previously then E.A2 comp-op v2.*

- **SIC Types Focusing on a Relationship as an Association.**

1. **Incidence Constraint:** allows a relationship occurrence to exist only if the participating entity occurrences exist.
2. **One_Side_Necessary_Condition for a Relationship Type:** restricts the existence of any occurrence(s) of the specified relationship type because there are conditions on one of its participating entity types. Different layouts of E-R diagrams (*line, star*, *loop-2*, or *loop-n* contexts) may have different conditions. Some examples include:

   - **Relationship_Depends_on_Relationship SIC:** requires that for a relationship occurrence to exist, one of its participating entity occurrences must participate in the other relationship type. It may happen in *line, star*, or *loop-2* contexts. For example, in a star context, Figure 5.3 (on page 111), we may have:
   if E RX F
   then E RY G
   or
   if E RX F
   then (E RY G) $\vee$ *(E RZ H)*

Note that stating "if *RX* then *RY*" is equivalent to stating "*RX* only-if *RY*". An example is "Customers (*F*) Use Products (*E*)" (i.e., *RX*) only-if "Factories (*G*) Make Products (*E*)"(i.e., *RY*). Sakai ([1983b]) states that *RX* is existentially dependent on *RY* and there is a hierarchical ordering of the entity types.

- **Relationship_Depends_on_Entity_Value SIC:** requires that for a relationship occurrence to exist, one of its participating entity occurrences must have some attribute value(s) satisfying a specified restriction. For example, in Figure 5.2, we may have
  if E RX F
  then (E.A1 comp_op1 v1) $\vee$ *(E.A2 comp_op2 v2)*
- **Exclusive_Relationship SIC:** requires that given two relationship types, for a relationship occurrence to exist, one of its participating entity occurrences must not participate in the other relationship type. These two relationship types *RX* and *RY* are exclusive (denoted as $RX \| RY$), and are called *excluding relationships* relative to the entity ([Palmer, 1978], [Kozaczynski and Lilien, 1988]). It may happen in any context. For example, in the context of Figure 5.2, we may have:
  if E RX F
  then $\neg$*(E RY G)*
  An example is where a *storeroom* is used either for storing *raw materials* or for storing *finished goods*, but never for both.
- **Not_And_Relationships SIC:** requires that given a group of relationship types with a sharing entity type, for a relationship occurrence to exist, its participating entity occurrence of the sharing entity type must not participate in all of the other relationship types. That is, no occurrence of the sharing entity type can participate in all the relationship types. It can only happen in a *star* context, e.g., in Figure 5.3, we may have:
  if E RX F
  then $\neg$*((E RY G)* $\wedge$ *(E RZ H))*

  If there are only two relationships involved, it reduces to an **Exclusive_Relationship SIC**.
- Some constraints are special because of temporal conditions:
  * **Relationship_Before_Relationship SIC:** requires that for a relationship occurrence to exist, one of its participating entity occurrences must participate in another relationship type at the time it is going to participate in this relationship type. For example, in Figure 5.2, we may have:
    if E RX F
    then E RY G before

It requires that *RY* must exist at the time *RX* is going to be inserted. However, there is no restriction on the deletion of *RY* after *RX* exists. A **Relationship_Depends_on_Relationship SIC** requires that when *RX* is inserted, *RY* exists (that is, *RX* and *RY* can be generated concurrently) and if *RY* is deleted, *RX* must be deleted too.

* **Entity_Attribute_Before_Relationship SIC:** requires that for a relationship occurrence to exist, the attributes of one of its participating entity occurrences must satisfy some conditions at the time it is going to participate in this relationship type. For example, in Figure 5.2, we may have:
if E RX F
then ((E.A1 comp_op v1) $\vee$ *(E.A2 comp_op2 v2)) before*

  An example is that "for a *person* to participate in a *life-insurance* relationship, his (her) *age* must be less than 65". Note that we do not care what attribute values this entity occurrence will have after it participates in the relationship.

* **Relationship_Not_Before_Relationship SIC:** requires that for a relationship occurrence to exist, one of its participating entity occurrences must not participate in another specific relationship type at the time it is going to participate in this relationship type.

3. **Two_Side_Necessary_Condition for a Relationship Type:** restricts the existence of any occurrence(s) of the specified relationship type because there are conditions on both participating entity types at the same time. Different layouts of E-R diagrams (*line, star, loop-2,* or *loop-n* contexts) may have different conditions. Some examples include:

   - **Pair_Relationships SIC:** requires that for a relationship occurrence to exist, if one of its participating entity occurrences participates or does not participate in some specific relationship type, the other participating entity occurrence must or must not participate in other specific relationship type(s). For example, in Figure 5.2, we may have:
   if E RX F
   then if E RY G
   then F RZ H
   - **Pair_Values SIC:** requires that for a relationship occurrence to exist, if one of its participating entity occurrences has certain attribute values, the other participating entity occurrence must have some specific attribute values. For example, in Figure 5.2, we may have:
   if E RX F
   then if (E.A comp_op1 v1)

then (F.B comp_op2 v2)

For example, *if Employee Is_Allocated Car then*
if Employee.Rank="president" then Car.Brand="BMW".

- **ID_Dependency_Relationship SIC:** requires that if an entity type E incorporates the primary key (e.g., F.Fkey) of the other entity type F as part of its candidate key (because of a relationship type implying an "ID dependency" or because of an *is_a*, or *component_of* special relationship type), for an occurrence of the relationship type to exist, its participating entity occurrences should have a condition: *E.Fkey=F.Fkey.*
- **Pair_Condition SIC:** requires that for a relationship occurrence to exist, there are some conditions involving the attributes of both of its participating entity occurrences. For example, in Figure 5.2, we may have:
if E RX F
then (E.A1 comp_op1 (F.B ari_op E.A2)), ...
- **Subset_Relationship SIC:** requires that for an occurrence of the specified relationship type to exist, its two participating entity occurrences must be connected via another relationship type ([Palmer, 1978]). That is, $RX \subseteq RY$, implying that RX occurrences are included in RY occurrences, or stated in a different way, RX only-if RY. It can only happen in a *loop-2* context. For example, *Owns* $\subseteq$ *Entitled-to-Drive* implies that if a *person* owns a *car*, he (she) must be entitled to drive it.
- **Exclusive_Occurrence SIC:** requires that for an occurrence of the specified relationship type to exist, its two participating entity occurrences must not be connected together via any occurrence of another relationship type ([MaFadden and Hoffer, 1988]). It is a special case, but weaker than **Exclusive_Relationship SIC**.
- **Relationships_Union Special SIC:** requires that for an occurrence of the specified relationship type to exist, its two entity occurrences must be connected together via any one of other relationship types. It can only happen in a *loop-2* context and is a necessary condition for the union of relationships, e.g., $RX = RY \cup RZ$. The semantics that one relationship (e.g., RX) is the union of the remaining relationships imply this type of SIC and **Subset_Relationship SIC**s (e.g., $RY \subseteq RX$, etc.).
- **Relationships_Intersection Special SIC:** requires that given a group of relationship types in a set *SuperRshipSet* and another relationship type *SubRship*, for each of the relationship types in *SuperRshipSet* to have an occurrence connecting the same pair of entity occurrences, there should be an *SubRship* occurrence connecting this pair of entity occurrences. It can only happen in a *loop-2* context and is a necessary condition for the intersection of relationships, e.g., $RX = RY \cap RZ$. The semantics that one relationship (e.g., RX) is the intersection of the remaining relationships

imply this type of SIC and **Subset_Relationship SICs** (e.g., $RX \subseteq RY$, etc.).

- **Relationships_Join SIC:** requires that if there is a linking path via some relationship types to connect two entity occurrences together, those two entity occurrences must be connected via another relationship type. It can only happen in a *loop_n* context and is a necessary condition for relationship composition, denoted by, e.g., $RX = RY \bowtie RZ$ ([Lenzerini and Santucci, 1983], [Azar and Pichat, 1987]). For example, in Figure 5.5, we may have:
  if (E RY H) ∧ (H RZ F)
  then E RX F
  In fact, this is another special case of **Pair_Relationships**.
- **Relationship_Depends_on_LoopN_Relationships SIC:** requires that for a relationship occurrence to exist, there is a linking path via other relationship types to connect its participating entity occurrences together. It can only happen in a *loop_n* context and is another necessary condition for the composition of relationships. For example, in Figure 5.5, we may have:
  if E RX F
  then (E RY H) ∧ (H RZ F)

  Sakai [1978] states that if RX is transitively dependent on RY and RZ and it lacks (nonkey) attributes, it is redundant and could be eliminated. However, Segev [1987] argues that some relationships that appear to be redundant may, in fact, carry semantic information and thus cannot be eliminated without losing information content, and even if it is redundant, there may be a choice of which relationships to eliminate and that this choice should be made during physical design.

4. **Intra_Condition for a Relationship Type:** restricts the existence of any occurrence(s) of the specified relationship type because of other occurrences of the same relationship type. For example, we may have the following.
   - **Symmetry and Transitivity Properties of a Relationship:** In the following, entity types E and F should be in a specialization hierarchy. A relationship type R is symmetric if the following condition is true: "if any occurrence *Occ1* of an entity type E is related via an R occurrence to an occurrence *Occ2* of the other entity type F, then an occurrence of the E type, corresponding to the same super-type occurrence of *Occ2*, is also related via another R occurrence to an occurrence of the F type, corresponding to the same super-type occurrence of *Occ1*." Some examples of such relationship types are: *Sibling_of, Married_to, Partner_of.*

     A relationship type R is transitive if the following condition is true: "if any occurrence *Occ1* of an entity type E is related via an R occurrence to

an occurrence *Occ2* of the other entity type F, and an occurrence of the E type, corresponding to the same super-type occurrence of *Occ2*, is related via a second R occurrence to an occurrence *Occ3* of the F type, then the *Occ1* is also related via a third R occurrence to the *Occ3*." Some examples of such relationship types are: *Sibling_of, Ancestor_of, Supervise, Partner_of.*

- **Relative Maximum Cardinality Constraint:** restricts the existence of a relationship occurrence because an occurrence of one its participating entity type can only participate in some maximum number of occurrences of the same relationship type.

  This restriction may only apply to a specified "implicit entity subtype" (it is a **Subtype_Relative_Maximum_Cardinality Constraint**). This restriction may become weaker because there are further conditions on the other entity type that relates to the specified entity type via the relationship type (it is a **Weaker_Relative_Maximum_Cardinality Constraint**). There may also be a SIC (that can be called **Subtype_Weaker_Relative_Maximum_Cardinality Constraint**) including both the complicated cases.

5. **Group_Relationships SIC:** requires that if occurrences of a group of relationship types exists, there must be some conditions on the values of attributes of those relationships or their sharing entity. For example, in Figure 5.2 or 5.3, we may have
   if (E RX F) ∧ (E RY G)
   then (E.A1 comp_op (RY.B1 ari_op RX.B2))
   Or, they may depend upon many other relationships, for example, we may have:
   if (E RX F) ∧ (E RY G)
   then (E RZ H) ∨ (E RS I) ∨ ...

- **Sufficient Conditions for the Existence of Relationship(s):** requires that given some conditions, one or more occurrences of the specified relationship type(s) must exist. It includes some special cases that are often mentioned in the literature.

1. **Totality Constraint:** requires that if an entity occurrence exists, it must participate in some minimum number of occurrences of the specified relationship type. A relationship type is total to one entity type if each occurrence of the entity type must participate in at least one relationship occurrence[106]. It is desirable to know an exact number (e.g., 1, 2, 5, etc.) as the relative minimum cardinalities for entity types if there exist such SICs. The minimum cardinality 0 is not a SIC.

[106] Some researchers use by other terms, e.g., [Palmer, 1982] states that the relationship is "mandatory", and [Kim et al., 1987] states that the relationship is "obligatory".

A relationship type may be only total to some specified "implicit entity subtype". That is a **Subtype_Totality Constraint**. A total constraint requirement may become stronger because there are some conditions on the other entity type that relates to the specified entity type (it is a **Stronger_Totality Constraint**). There may also be a SIC (that can be called **Subtype_Stronger_Totality Constraint**) including both complicating cases.

There may also some further restrictions on the relationship occurrences. For example, a **Weak_Entity SIC** requires that a relationship type, via which the weak entity type is dependent upon a regular entity type, is total to the weak entity type and its key is fixed. A **Critical_Relationship_Occurrence SIC** requires the totality constraint to the specified entity type and the existence of exactly one critical relationship occurrence.

2. **Completeness_Mapping SIC:** requires that if an entity occurrence exists, it must be related to all occurrences of the other entity type via the specified relationship type ([Webre, 1983]).

   A relationship type may be only complete to some specified "implicit entity subtype". That is, **Subtype_Completeness_Mapping SIC**. A complete mapping requirement may also become weaker because there are some conditions on the other entity type that relates to the specified entity type (it is a **Weaker_Completeness_Mapping SIC**). There may also be a SIC (that can be called **Subtype_Weaker_Completeness_Mapping SIC**) including both complicating cases.

3. **Either_Existence_Relationship SIC:** requires that if one entity occurrence exists, it should participate in at least one (or some specified number of) occurrence(s) among a group of relationship types. For example, in a star context, Figure 5.3, each E occurrence may be required to participate in at least one relationship occurrence among the relationship types RX, RY and RZ.

   There are some variants of this type, e.g., the cases considering an "implicit entity subtype", different quantitative requirements on relationship occurrences, or having further restrictions. For example, in Figure 5.3, each E occurrence may be required to participate in at least either one RX occurrence, two RY occurrences or four RZ occurrences.

4. **Relationship_Trigger_Relationship SIC:** requires that if some relationships existed in the past (and no longer exist now), other relationships must exist. For example, in Figure 5.2, we may have:
   if E RY G previously
   then E RX F

5. **Entity_Value_Trigger_Relationship SIC:** requires that if the value(s) some attribute(s) of an entity occurrence satisfied a certain condition in the past

(and no longer satisfy now), the entity occurrence must participate in some minimum number of occurrences of the specified relationship type. For example, we may have:
if E.A1 comp-op v1 previously
then E RX F

- **SIC Types for Entities without Explicit Relationships.**
  - **SIC Types for Entities in a Specialization Hierarchy.**
    1. **Exclusion between Entity Types:** requires two entity types to be exclusive.
    2. **Entities_Intersection Special SIC:** requires that given a group of entity types *E1, E2, ..., En* having a common candidate key *Ekey*, for each of *E1, E2, ... En_1* entity types to have any occurrence with the same candidate key value, there should be an occurrence with this candidate key value in the specified entity type *En*. That is, *"if (E1.Ekey=Value)* $\wedge$ *(E2.Ekey=Value)* $\wedge$ *... then En.Enkey=Value"*. It is a necessary condition for $En = E1 \cap E2 \cap \ldots$.
    3. **Entities_Union Special SIC:** requires that given a group of entity types *E1, E2, ..., En* having a common candidate key *Ekey*, for any occurrence in the specified entity type *En*, there should be at least one corresponding occurrence with the same candidate key value in one of other entity types. That is, *"if E.Ekey=Value then (E1.Ekey=Value)* $\vee$ *(E2.Ekey=Value)* $\vee$ *... "*. It is a necessary condition for $En = E1 \cup E2 \cup \ldots$
  - **Association Abstraction SIC:** requires that a set occurrence cannot be deleted if it is not empty; two kinds of derived attributes in a set — the indexing attribute and the key of the indexing entity type — cannot be updated; and some formulas involving aggregate functions on attribute(s) of the member entity type although there is no implicit relationship type *member_of*.

Appendix E

Examples of Heuristics

This appendix presents some heuristics that can be used to reduce the effort required to use the *SIC elicitation subsystem.* The listing shows only some examples; it is far from complete.

1. Two heuristics may be applied to elicit old_new transitional constraints.
   - If an attribute is numeric, the subsystem may inquire whether it is monotonically increasing or decreasing on update.
   - If an attribute has an enumerated value set, there may be update transitions that are not allowed. For example, the value of *Marriage* status cannot be updated from "single" to "widowed", or "divorced", nor conversely.
2. An attribute with the "date" data type may be unchangeable. More precisely, we should state that historical dates are usually unchangeable. However, the subsystem needs to know the meaning of the date in order to know whether it is historical.
3. Based on domains of attributes, the subsystem may suggest some possible entity SICs, e.g., **Formula SICs**, to the database designer. For example, if an entity has several attributes with the same domain *"money"*, there may be a formula SIC involving these attributes. However, without application domain knowledge, the subsystem would have to ask the database designer to confirm whether such a constraint is needed and provide the necessary details.
4. The subsystem may have a lexicon containing the names of common relationship types with symmetry or transitivity property. However, the subsystem still needs to ask the database designer to confirm it. For example, the relationship *"Married_to"* is usually symmetric. However, if it is between two entity types *Woman* and *Man* (or *Wife* and *Husband*), it would not be symmetric. Another example is that the relationship *Supervise* (or *Manage*) is usually transitive. However, if the database designer in fact defines the relationship to be *"directly supervise (or manage)"*, it would not be transitive.
5. A heuristic to detect excluding relationships or time sequencing between relationships is:

If two relationships, which are adjacent in an E-R diagram, are named by the database designer using the same verb phrase, they usually imply exclusion or even time sequence among them. For example, the names of the relationships, *Customer_Owns_Car, Dealer_Owns_Car* are created by using the same verb phrase *"Owns"*.
If those relationships (e.g., *Customer_Owns_SavingAccount* and *Customer_Owns_CheckingAccount*) are not exclusive nor have time sequencing, it may be the following case.

6. A heuristic to detect a **Group_Relationships SIC** is:
Suppose that an entity type is connected to a group of relationship types. Examine attribute names, but omit their prefixed entity or relationship names. If there is a common attribute name other than *Id* or *Name* among either (1) the sharing entity type and these relationship types or (2) the sharing entity type and its related other entity types, it is likely that there is a formula between those attributes. An example is *Inventory.Qty= sum(Supply.Qty) − sum(Sales.Qty)* in the context of two relationship types, *Supply* and *Sales*, sharing the same entity type, *Inventory*. Another example is *Customer.AccountBalance = SavingAccount.Balance + CheckingAccount.Balance* in the context of a *Customer* owning two entity types, *SavingAccount* and *CheckingAccount*.

Appendix F

Verification of Aggregate Attribute SICs and Cardinalities

This appendix discusses the extent to which we can verify SICs involving aggregate attributes, and provides algorithms to verify absolute and relative cardinalities given by the database designer.

F.1 Simple Tests on Aggregate Attribute SICs

Suppose that we have some aggregate attribute SICs, each of which is a simple assertion to restrict the aggregate value of an attribute. At best, we can only have some simple tests, such as:

> Suppose *E.A* is an attribute between *v1* and *v2*; *avg(E.A)* is specified to be between *MinAvg* and *MaxAvg*; *sum(E.A)* is specified to be between *MinSum* and *MaxSum*; the maximum absolute cardinality of *E* is specified as *AbsCmax*. The following should be true:
> $v1 \leq MinAvg \leq MaxAvg \leq v2$;
> $v1 \leq MinAvg \leq MinSum \leq MaxSum \leq (AbsCmax \times MaxAvg) \leq (AbsCmax \times v2)$.

> If there is a **Subtype_Aggregate_Attribute SIC** on *E.A*, the following should be true:
> $v1 \leq$ *subtype's min* $\leq$ *subtype's max* $\leq v2$;
> *subtype's MaxSum* $<$ *MaxSum*;
> *subtype's count(E)* $<$ *AbsCmax*

The general **Interdependent_Aggregate_Attributes, Aggregate_Attribute_Formula, Subtype_Aggregate_Attribute_Formula**, etc. cannot be verified conceptually.

F.2 Algorithms for Verifying Cardinalities

1. **Verification of "traditional" relationship cardinalities:**

   - A relative minimum cardinality of 0 is not a constraint. Therefore, if a SIC exists, for each involved entity type in a relationship, its relative cardinalities must be:
     $1 \leq$ *relative minimum cardinality* $\leq$ *relative maximum cardinality.*
   - Suppose that the relative maximum cardinality of entity type E through a relationship R to entity type F is *Cmax*, and absolute maximum cardinality of F is *Ab_max*. Then, $Cmax \leq Ab\_max$.
   - A set of cardinality constraints may be inconsistent in the *loop-2* and *loop-n* contexts.
     Given a group of relationship and entity types with their relative cardinalities, do the following check.
     (a) Associate each entity type *Ei* with a special variable *Ei#*.
     (b) For each relationship type *Ri*, do the following. Suppose that *Ri* connects two entity types *Ej*, *Ek*; and the minimum and maximum cardinalities of *Ej* and *Ek* in *Ri* are, respectively, *(Cminj,Cmaxj)* and *(Cmink,Cmaxk)*. That is,

     Ej(Cminj,Cmaxj) Ri Ek(Cmink,Cmaxk).

     - If *Cmaxj* is not "*" and *Cmink* is not 0, construct the following inequality:
       $Cmaxj \times Ej\# \geq Cmink \times Ek\#$
     - If *Cmaxk* is not "*" and *Cminj* is not 0, construct the following inequality:
       $Cmaxk \times Ek\# \geq Cminj \times Ej\#$

     (c) Solve all the above inequalities for the special variables *Ei#*'s under the restriction of each $Ei\# > 0$.
     (d) If there is no solution, the cardinalities are inconsistent[107].

[107] Observe these inequalities. If the cardinalities are consistent, the designed database can be populated. Then there is at least one solution for these inequalities: assigning to each *Ei#* the value corresponding to the number of occurrences of *Ei* in the database. Formally, Lenzerini and Nobili ([1987]) have proved that there exist solutions for all those inequalities if and only if the cardinalities are consistent. Note that their inequalities contain another set of variables *R#* because they prove that it is true for any degree of relationships (not restricted to binary relationships). However, in the case of

- Cardinality constraints can interact with other SICs. A special interaction is with a **Subset_SIC**. Suppose we have relationship type *"RX ⊆ RY"* between the entity types, *E* and *F*. Then, for either *E* or *F*, both minimum cardinality and maximum cardinality in *RX* should be less than or equal to those in *RY*, respectively. If either *E* or *F* has (1,1) in both *RX* and *RY*, *RX* and *RY* would contain the exactly same occurrences[108]. They should be merged and a new relationship type will be created.
- **Verification of Completeness_Mapping SIC:** Suppose that a relationship type *R* is specified to be complete relative to an entity type *E*. Certainly it is also complete relative to its other entity type *F* by the symmetry of this SIC. The relative maximum cardinality of *E* in *R* must be equal to the absolute maximum cardinality of *F*. Otherwise, they are inconsistent. There is a similar condition between the relative maximum cardinality of *F* and absolute maximum cardinality of *E*.

2. **Verification of Stronger, Weaker, and Sub-type Cardinalities:** When we consider cardinalities with regard to **Subtype_Totality Constraint** (*Sub_Cmin*), **Subtype_Relative_Maximum_Cardinality Constraint** (*Sub_Cmax*), **Stronger_Totality Constraint** (*Stronger_Cmin*), **Weaker_Relative_Maximum_Cardinality Constraint** (*Weaker_Cmax*), **Subtype_Stronger_Totality Constraint** (*Sub_Stronger_Cmin*), and **Subtype_Weaker_Relative_Maximum_Cardinality Constraint** (*Sub_Weaker_Cmax*), verification becomes complicated.

   - The following should be true if these cardinality constraints are not redundant and are consistent with each other.
     - For the same "subtype",
       $1 \leq Cmin < Sub\_Cmin \leq Sub\_Cmax < Cmax$
     - For the same "further restriction",
       $1 \leq Stronger\_Cmin \leq Weaker\_Cmax$,
       $Stronger\_Cmin < Cmin$,
       $Weaker\_Cmax < Cmax$.

---

binary relationships, if we can assure that each pair of the relative maximum cardinality is greater than or equal to relative minimum cardinality, those inequalities containing the *R#* variables can be reduced to our inequalities. Lenzerini and Nobili have also proposed another way to detect inconsistency for cardinalities by discovering cycles with special "weights" in an E-R diagram.

[108] Prove it by contradiction as follows. We already know that *RX* ⊆ *RY*. Now suppose that entity type *E* has (1,1) in both relationship types, *RX* and *RY*. If an *RY* occurrence, via which an occurrence *e1* of *E* type connects with an occurrence *f1* of *F* type, does not belong to *RX* type, then *e1* must connect another occurrence *f2* of *F* type via the relationship type *RX*. Otherwise, it would violate the totality constraint of *RX* to *E*. Then, by *RX* ⊆ *RY*, *e1* must also connect with *f2* via the relationship type *RY*. It would violate the maximum constraint of *RY* relative to *E* unless *f1* = *f2*. So, we would also have *RY* ⊆ *RX*.

- For the same "subtype", and the same "further restriction",
  $Stronger\_Cmin < Sub\_Stronger\_Cmin \leq Sub\_Weaker\_Cmax < Weaker\_Cmax$,
  $Sub\_Stronger\_Cmin < Sub\_Cmin$,
  $Sub\_Weaker\_Cmax < Sub\_Cmax$.

- In the contexts of *loop-2, loop-n*, if cardinalities have been specified for "implicit subtype(s)", the Lenzerini and Nobili's inequalities should be tested by using the proper cardinality values.
  - For example, in the context of *loop-2* where two relationship types RX and RY exist between two entity types E and F, the database designer may consider an implicit subtype of E, and give the *Sub_Cmin, Sub_Cmax* of RX and RY relative to E, *Stronger_Cmin, Stronger_Cmax* of RX and RY relative to F. There may be potential inconsistencies between these cardinalities.
  - Another example in the context of *loop-n* has a loop linking entity types E, F, G and H, through four relationship types RX, RY, RZ and RS, consecutively. The database designer may consider implicit subtypes of adjacent entity types, E and F. Now the proper values for testing the above inequalities would be:

    Sub_Stronger_Cmin, Sub_Weaker_Cmax of RX relative to E and F,
    Sub_Cmin, Sub_Cmax of RY relative to F,
    Stronger_Cmin, Weaker_Cmax of RY relative to G,
    Cmin, Cmax of RZ relative to G, and H,
    Stronger_Cmin, Weaker_Cmax of RS relative to H,
    Sub_Cmin, Sub_Cmax of RS relative to E.

Appendix G

Consistency and Nonredundancy Rules for SIC Elicitation Subsystem

The following rules could be stored in an *elicitation subsystem* for capturing SICs. These rules are used to expedite the elicitation and verification procedure. By using them, the subsystem could sometimes avoid the need to invoke a sophisticated logic verification algorithm. In other cases, they eliminate the need to ask the database designer to confirm some SICs that are "obviously" inconsistent or redundant. The listing here is illustrative rather than complete. In the following, *E, F, G, H, I, ...* are used to denote entity types. The *R, R1, R2, R3, ...* are used to denote relationship types.

1. If an attribute is declared to be non-changeable (i.e., a **Nonvolatility constraint**) there should be no other update SICs asserted for it.

2. A composite key constraint is consistent with domain constraints if each component attribute of a composite key is declared to be not-null.

3. Incidence constraints are always needed and unlikely to be inconsistent with other SICs. We need not verify them for consistency and non-redundancy.

4. A **Relationship_Depends_on_Relationship SIC** could be redundant or could be reduced to another SIC if any involved relationship is total. If there is a negative condition asserted for the involved relationships, specifying this type of SIC would be inconsistent.

   - If *R1* is total to *E*, the SIC, *"if E R1 F then E R2 G"*, is subsumed by the totality constraint — *R2* total to *E*. On the other hand, if *R2* is total to *E*, this SIC is trivially true.
   - The SIC relative to *E*, *"if E R1 F then E R2 G"*, or *"if E R2 G then E R1 F"*, is inconsistent with an **Exclusive_Relationship SIC**, $R1 \| R2$, that is *"if E R1 F then* $\neg$*(E R2 G)"*.
   - If we have two SICs relative to *E*, *"if E R1 F then E R2 G"* and *"if E R1 F then E R3 H"*, they are inconsistent with either an **Exclusive_Relationship SIC** — $R2 \| R3$, or a **Not_And_Relationship SIC** — "$\neg((E\ R1\ F) \wedge (E\ R2\ G) \wedge (E\ R3\ H))$".

- If we have two SICs, *"if E R1 F then E R2 G"* and *"E R2 G then E R3 H"*, they are inconsistent with a **Not_And_Relationship SIC**, "$\neg$*((E R1 F)* $\wedge$ *(E R2 G)* $\wedge$ *(E R3 H))*".
- Both *"if with_respect_to E, E R1 F then with_respect_to E, E R2 F"* and *"if with_respect_to F, E R1 F then with_respect_to F, E R2 F"* become redundant if a **Subset_Relationship SIC**, $R1 \subseteq R2$, i.e., *"if E R1 F then E R2 F"* has been specified.
- If we have further restrictions, they may be redundant because they may be implied by another SIC. For example, the further restriction *"(F R3 H)"* in the SIC, *"if E R2 G then (E R1 F)* $\wedge$ *(F R3 H)"*, is redundant if there is another SIC to require *"if E R1 F then F R3 H"*.
- If *R1* is total to *E*, the SIC, *"if E R1 F then (E R2 G)* $\vee$ *(E R3 H)"*, reduces to an **Either_Existence_Relationship SIC** *"(E R2 G)* $\vee$ *(E R3 H)"*. This SIC is also subsumed by either of the following:
  — *R2 total to E (i.e.,* $\forall$ E, $\exists$ R2, E R2 G*)*,
  — *R3 total to E (i.e.,* $\forall$ E, $\exists$ R3, E R3 H*)*,
  — *"if E R1 F then E R2 G"*,
  — *"if E R1 F then E R3 H"*.
  This SIC is inconsistent with a set of two **Exclusive_Relationship SICs** — $R1 \| R2$ and $R1 \| R3$.

5. An **Exclusive_Relationship SIC**, $R1 \| R2$, is inconsistent with the set of totality constraints — *both R1 and R2 are total to E*. If $R1 \| R2$ and only one (say *R1*) of the relationships is total to *E*, the database designer needs to disconnect the other relationship (i.e., *R2*) type from *E* since it becomes unrelated to *E*; otherwise, they are inconsistent.

6. An **Exclusive_Occurrence SIC** becomes redundant when an **Exclusive_Relationship SIC** has been specified. However, it can exist when the set of totality constraints, *both R1 and R2 are total to E*, has been specified.

7. A **Not_And_Relationships SIC**, *"*$\neg$*((E R1 F)* $\wedge$ *(E R2 G)* $\wedge$ *(E R3 H))"*, becomes redundant when any pair of the involved relationships is exclusive, i.e., $R1 \| R2$, $R2 \| R3$, or $R3 \| R1$. It is inconsistent with the set of totality constraints — *all involved relationships* R1, R2, *and* R3 *are total to* E. If only one relationship, say *R3*, is total, it reduces to a **Not_And_Relationship SIC** among the remaining ones (or an **Exclusive_Relationship SIC** if there are two remaining). It is not meaningful that only one of the relationship types (say, *R3*) is not total since "$\neg R3$" would imply that the *R3* relationship type should be disconnected.

8. An **Either_Existence_Relationship SIC**, *"(E R1 F)* $\vee$ *(E R2 G)* $\vee$ *(E R3 H)"*, becomes redundant when any one of these involved relationship is total to *E*. It is also redundant when a proper subset of these relationships has been declared to

have an **Either_Existence_Relationship SIC**. A **Subtype_Either_Existence Relationship SIC**, *"if (E.A1 comp_op v1) then (E R1 F) ∨ (E R2 G) ∨ (E R3 H)"*, becomes redundant when the related *sub-Cmin* (the subtype minimum cardinality) of any one of these involved relationships to *E* is greater than or equal to 1. It is inconsistent with the set of three **Relationship_Depends_on_Entity_Value** SICs:
"if E R1 F then ¬ (E.A1 comp_op v1)",
"if E R2 G then ¬ (E.A1 comp_op v1)",
and *"if E R3 H then ¬ (E.A1 comp_op v1)"*.

9. A **Pair_Relationships SIC**, *"if (E R1 F) then if ((E R2 G) then (F R3 H))"*, is redundant when *R3* is total to *F*. When *R2* is total to *E*, it reduces to a **Relationship_Depends_on_relationship SIC**, *"if (E R1 F) then (F R3 H)"*. This SIC may be redundant or needs to be modified when there are some **Relationship_Depends_on_Relationship SICs**. For example, if we have "*if (E R1 F) then (F R3 H)*", the above SIC becomes redundant; if we have "*if (E R1 F) then (E R2 G)*", the above SIC reduces to "*if (E R1 F) then (F R3 H)*". However, in case that there are further restrictions on *R2* and *R3*, the consistency needs to be checked during the consultation and is also hard to verify.

10. A **Relationship_Before_Relationship SIC**, "*if E R1 F then (E R2 G) before*", is redundant when *R2* is total to *E*. It is not meaningful when *R1* is total to E. If the database designer specifies a time sequence cycle among some relationships (e.g., *"if E R1 F then (E R2 G) before"*, *"if E R2 G then (E R3 H) before"*, *"if E R3 H then (E R1 F) before"*), the insertion of all of the relationships can only performed in a single transaction. It is possible that the database designer has made a mistake to assert the logical meaning of these relationship types. This type of SIC can exist when either a **Relationship_Depends_on_Relationship**, **Subset_Relationship SIC**, or an **Exclusive_Relationship SIC**, etc. has been specified for the related relationships *R1* and *R2*.

11. A **Relationship_Not_Before_Relationship**, *"if (E R1 F) then ¬ (E Rn Y) before"* is inconsistent with the totality constraint that *Rn* is total to *E*, or *"if E R1 F then (E Rn Y) before"*, or a set of SICs: *"if E R1 F then (E R2 G) before"*, *"(E R2 G) then (E R3 H) before"*, *"if E R3 H . . . "*, *"if E Ri X then E Rn Y before"*. It is not meaningful when *R1* is total to *E*. However, it is allowed to exist when either a **Relationship_Depends_on_Relationship**, **Subset_Relationship SIC**, or an **Exclusive_Relationship SIC**, etc. has been specified for the related relationships *R1* and *R2*.

12. If a **Group_Relationships SIC** is specified, the verification is complicated.

    - The SIC, *"if (E R1 F) ∧ (E R2 G) then (E R3 H)"*, is redundant when *R3* is total to *E*. It reduces to *"if (E R2 G) then (E R3 H)"* when *R1* is total to *E*. In

addition, when both *R1* and *R2* are total to *E*, it is subsumed by a totality constraint — *R3* is total to *E*. It is inconsistent with a **Not_And_Relationships SIC**, "$\neg$*((E R1 F) $\wedge$ (E R2 G) $\wedge$ (E R3 H))*".

- Two SICs, *"if (E R1 F) $\wedge$ (E R2 G) then E R3 H"* and *"if (E R1 F) $\wedge$ (E R2 G) then E R4 I"*, are inconsistent with $R3 \| R4$, or "$\neg$*((E R1 F) $\wedge$ (E R2 G) $\wedge$ (E R3 H) $\wedge$ (E R4 I))*".
- Two SICs, *"if (E R1 F) $\wedge$ (E R2 G) then (E R5 J)"* and *"if E R5 J then E R6 K"*, are inconsistent with "$\neg$*((E R1 F) $\wedge$ (E R2 G) $\wedge$ (E R5 J) $\wedge$ (E R6 K))*".
- If this type of SIC has the disjunction of assertions in the "then" part, it would be further complicated and needs a case analysis to verify consistency.
  - The SIC, *"if (E R1 F) $\wedge$ (E R2 G) then (E R3 H) $\vee$ (E R7 L)"*, is inconsistent with a set of two **Not_And_Relationships SICs**: "$\neg$*((E R1 F) $\wedge$ (E R2 G) $\wedge$ (E R3 H))*" and "$\neg$*((E R1 F) $\wedge$ (E R2 G) $\wedge$ (E R7 L))*".
  - Two SICs, *"if (E R1 F) $\wedge$ (E R2 G) then (E R3 H) $\vee$ (E R7 L)"*, and *"if (E R1 F) $\wedge$ (E R2 G) then (E R8 M) $\vee$ (E R9 N)"* are inconsistent with either a set of four exclusive relationships: $R3 \| R8, R3 \| R9, R7 \| R8, R7 \| R9$; or a set of four **Not-And-Relationships SICs**:
    "$\neg$*((E R1 F) $\wedge$ (E R2 G) $\wedge$ (E R3 H) $\wedge$ (E R8 M))*",
    "$\neg$*((E R1 F) $\wedge$ (E R2 G) $\wedge$ (E R3 H) $\wedge$ (E R9 N))*",
    "$\neg$*((E R1 F) $\wedge$ (E R2 G) $\wedge$ (E R7 L) $\wedge$ (E R8 M))*"
    and "$\neg$*((E R1 F) $\wedge$ (E R2 G) $\wedge$ (E R7 L) $\wedge$ (E R9 N))*".
  - Similarly, two SICs, *"if (E R1 F) $\wedge$ (E R2 G) then (E R3 H) $\vee$ (E R7 L)"*, and *"if (E R1 F) $\wedge$ (E R2 G) then (E R9 N)"* are inconsistent with either a set of two exclusive relationships: $R3 \| R9, R7 \| R9$; or a set of two **Not_And_Relationships SICs**:
    "$\neg$*((E R1 F) $\wedge$ (E R2 G) $\wedge$ (E R3 H) $\wedge$ (E R9 N))*",
    and "$\neg$*((E R1 F) $\wedge$ (E R2 G) $\wedge$ (E R7 L) $\wedge$ (E R9 N))*".
  - Three SICs, *"if (E R1 F) $\wedge$ (E R2 G) then (E R3 H) $\vee$ (E R7 L)"*, *"if E R3 H then E R9 N"*, and *"if E R7 L then E R5 J"*, are inconsistent with a set of two **Not_And_Relationships**:
    "$\neg$*((E R1 F) $\wedge$ (E R2 G) $\wedge$ (E R3 H) $\wedge$ (E R9 N))*",
    and "$\neg$*((E R1 F) $\wedge$ (E R2 G) $\wedge$ (E R7 L) $\wedge$ (E R5 J))*".

13. A **Relationships_Join SIC**, *"if (E R1 F) then (E R2 H) $\wedge$ (H R3 F)"*, is not redundant even if either *R2* or *R3* is total to the related entities. It is inconsistent with the **Exclusive_Relationship SIC** among any pair of *R1, R2, R3.*
    If we have this SIC, two **Relationship_Depends_on_Relationship SICs**, *"if (E R1 F) then (E R2 H)"* and *"if (E R1 F) then (H R3 F)"*, are redundant.

14. The **Symmetry, and Transitivity Properties** of a relationship would not be inconsistent with other SICs in the above.

15. If *R1* is total to *E*, a **Relationship_Depends_on_Entity_Value SIC**, *"if (E R1 F) then (E.A comp_op v)"* reduces to *"(E.A comp_op v)"*.

16. A **Weak Relationship SIC** or a **weak entity type** requires the presence of the specification totality constraints.

Appendix H

SIC Reformulation and Decomposition Algorithms

This appendix proposes algorithms for reformulating general SICs in the simplified format of Appendix C and decomposing them, if necessary, into operation-dependent sub-SICs as defined by the Representation model. Before applying the algorithms, the subsystem takes the responsibility of writing all general SICs in the simplified format of Appendix C. The *if ... then ...* rule format should be used if possible. For example, although the format "$\neg P \vee Q$" is equivalent to "*if P then Q*" in logic, the rule format should be used. If a general SIC is originally written by using a nested rule (i.e., *if ... then if ... then ...*), it should be rewritten by using only one pair of *"if"* and *"then"* keywords. Decomposed sub-SICs have the same certainty factor as their general SICs.

H.1 Find the Relevant Object and Operation Components

1. If the system variable *Current_time* appears in a general SIC, a sub-SIC for *Current_time* on update will usually be required. However, there are two following exceptions because the increase of *Current_time* will never violate the SIC.
   (a) The SIC is in rule format and *Current_time* appears on the left hand side of "$\leq$" in the *"if"* part.
   (b) The SIC is not in rule format (or the SIC is in rule format but *Current_time* is in the *"then"* part), and *Current_time* appears on the left hand side of "$\geq$".
2. If an attribute (say *E.A*) is mentioned in a SIC, the SIC is usually relevant to it on update. However, if any of the following cases occurs, the SIC would <u>not</u> be relevant to it.
   (a) The attribute has been declared to be unchangeable (i.e., there is a SIC *"if* E.A *is_to_be_updated then false"*).
   (b) The SIC contains a pair of *new* and *old* special functions with arguments of other attributes, not this attribute.
   (c) The SIC contains the keyword *is_to_be_deleted.*
   (d) The attribute is in the *"then"* part and the modifier "*before*" is attached to it.

(e) The attribute is in the *"then"* part and there is a modifier *"previously"* in the *"if"* part.

(f) The attribute is a candidate key and is referenced by a special formula in the *"if"* part of the SIC. This formula involves the same attribute name of another entity type (e.g., $E.A{=}F.A$) to link the specific mentioned entity type occurrences (e.g., E and F) that corresponds to the same physical entity occurrence (e.g., G) in a specialization hierarchy.

3. Suppose that a general SIC has been found by the above steps to be relevant to an attribute.

(a) Suppose that the relevance of the SIC to the attribute is <u>not</u> because the attribute is an argument of an aggregation function. In general, the SIC is also relevant to its associated entity or relationship on insertion. However, if any of the following cases occurs, the SIC would <u>not</u> be relevant to the associated entity or relationship on insertion.

i. The SIC is the one that declares the attribute to be unchangeable.

ii. The SIC contains a pair of *new* and *old* functions.

iii. The SIC contains an *"previously"* modifier in its *"if"* part or a *"before"* modifier in its *"then"* part.

In addition, if the attribute belongs to an entity and the SIC is also relevant to the insertion of one mentioned relationship in which the entity participate, the SIC is <u>not</u> relevant to the entity on insertion.

(b) Suppose that the SIC is relevant to the attribute because the attribute is an argument of an aggregation function. In general, the SIC is relevant to its associated entity or relationship on both insertion and deletion. However, if the assertion including the attribute is in the form of *"agg_fcn(E.A) comp_op arithmetic_simple_expression"*, where *"agg_fcn"* is an aggregate function and *"comp_op"* is a comparison operator (for a *comp_op* (e.g., ">") prefixed with a *"not"* or "¬", first properly replace the comparison operator (e.g., "≤")), the following would be exceptions:

i. Suppose that the aggregate function is either *max* or *count*:

- If the comparison operator is either ">, ≥", the SIC is only relevant to the mentioned entity/relationship on deletion, not insertion.
- If the comparison operator is either "<, ≤", the SIC is only relevant to the mentioned entity/relationship on insertion, not deletion.

ii. Suppose that the aggregate function is *min*:

- If the comparison operator is either ">, ≥", the SIC is only relevant to the mentioned entity/relationship on insertion, not deletion.
- If the comparison operator is either "<, ≤", the SIC is only relevant to the mentioned entity/relationship on deletion, not insertion.

iii. If the aggregate function is *sum* and if it can be assumed that the attribute values to be summed are all positive, the sub-SICs are the same as the case of either *max* or *count*.

In all other cases (e.g., the comparison operator is "=" or "≠", or the aggregate function is *"avg"*), the SIC is relevant to its associated entity or relationship on both insertion and deletion. However, similarly, if the attribute belongs to an entity and the SIC is also relevant to the insertion of one mentioned relationship in which the entity participate, it is not relevant to the entity on insertion.

4. Suppose that a SIC contains the keyword *"is_to_be_deleted"*. The SIC is only relevant to the entity or relationship that is to be deleted.

5. Suppose that a SIC does not contain the keyword *"is_to_be_deleted"*.

   (a) Consider its mentioned relationships.

      i. Suppose that in the *"if"* part of a SIC there is an assertion containing a relationship (say R). In general, the SIC is relevant to the relationship on insertion. However, if any of the following cases occurs, the SIC would be relevant to it on deletion, not insertion.

         - A *"not"* or "¬" has been attached to the assertion.
         - An "∃ *at_most*" numerical quantifier has been attached to the relationship R or there is a *"count(R)* $<$ *Number"* or *"count(R)* $\leq$ *Number"* assertion, but no *"not"* or "¬".
         - A modifier *"previously"* has been attached to the assertion.

      If there is an "∃ exactly" numerical quantifier or *"count(R)=Number"* assertion, the SIC would be relevant to the relationship on both insertion and deletion.

      ii. Suppose that the *"then"* part of a SIC or a SIC, which is not in rule format, contains an assertion referencing to a relationship. In general, the SIC is relevant to the relationship on deletion. However, if there is a *"before"* modifier attached to it, or if there is an *"previously"* modifier appearing in the *"if"* part of the SIC, this SIC is not relevant to it. In addition, if any of the following cases occurs, the SIC would be relevant to it on insertion, not deletion.

         - A *"not"* or "¬" has been attached to the assertion.
         - An "∃ *at_most*" numerical quantifier has been attached to the relationship or there is a *"count(R)* $<$ *Number"* or *"count(R)* $\leq$ *Number"* assertion, but no *"not"* or "¬".
         - A "∀" quantifier has been attached to the relationship, but no *"not"* or "¬".

If there is an "$\exists$ exactly" numerical quantifier or *"count(R)=Number"* assertion, the SIC would be relevant to the relationship on both insertion and deletion.

(b) Consider its mentioned entities. If there are relationships involved in the form of *"entity_type1 relationship_type entity_type2"* of a SIC, do not consider the non-sharing entities. The *sharing entities* are those entities which:
— appear in at least two assertions as a participant of relationships or as an owner of an attribute in either *"if"* or *"then"* part if there is no "*with_respect_to*" modifier; or
— are only those in the "*with_respect_to*" modifiers.
For example, in the case of "*if (E RX F) then (E RY G)*", E is a sharing entity, F and G are not. In the case of *"if (E RX F) then (E RY H)* $\wedge$ *(F RZ H)"*, all entities E, F, and H are sharing entities. In the case of "*if with_respect_to E, (E RX F) then with_respect_to E, (E RY F)*", E is a sharing entity, but F is not. If those involved entity types are in the same specialization hierarchy, the sharing entity types may be different in syntax. For example, assume that some mangers do not supervise employees, we may have
"*if with_respect_to Manager, (Manager Supervise Employee)*
then with_respect_to Employee, $\neg$ *(Employee Participate Union)*"
It means that if a manager supervises employee(s), he (she) cannot, as an employee, participate in a union. Here the "sharing" entity type *Manager* in the relationship *Supervise* corresponds to *Employee* in the relationship *Participate.* A more complicated case would need a list of sharing entity types in some order and their corresponding sharing entity types in the same order. For example,
"if with_respect_to (Teacher, Student), (Teacher Instruct Student)
then with_respect_to (Worker, Manager), $\neg$ *(Manager Supervise Worker)* "
This is an exclusive occurrence example which means that "if a teacher instructs a student, the student cannot, as a manager, supervise the teacher as a worker" (but the other way around may be permitted). Here the sharing entity types are: *Teacher* corresponding to *Worker*, and *Student* corresponding to *Manager.*

i. Suppose that in the *"if"* part of a SIC there is an assertion referencing to an entity. In general, the SIC is relevant to the entity on insertion. However, if the SIC is relevant to the insertion of a relationship in which the entity participates, the SIC is <u>not</u> relevant to the entity on insertion.

ii. Suppose that the *"then"* part of a SIC or a SIC, which is not in rule format, contains an assertion referencing to an entity. In general, if the assertion is not on its attribute(s), the SIC is relevant to the entity on deletion. If any of the following cases occurs, the SIC would be relevant to it on <u>insertion</u>, not deletion. However, if the SIC is relevant to the insertion of a relationship in which the entity participate, it is not relevant to the

insertion of the entity; similarly, if the SIC is relevant to the deletion of a relationship in which the entity participate, it is not relevant to the deletion of the entity.

- A *"not"* or "¬" has been attached to the assertion.
- An *"∃ at_most"* numerical quantifier has been attached to the entity or there is a *"count(R) < Number"* or *"count(R) ≤ Number"* assertion, but no *"not"* or "¬".
- A *"∀"* quantifier has been attached to the entity, but no *"not"* or "¬".

(c) If there is a SIC including an aggregate function because of the natural association or indexing derived set association, the SIC is also relevant to the entity, as a set object, on deletion.

6. Suppose that by the above steps, a general SIC is found to be relevant to the insertion or deletion of a relationship R or entity E. If its primary key is also explicitly mentioned in the SIC, skip this step. Otherwise, do the following.

   (a) Suppose that the SIC only contains attributes of the relationship type R or entity type E. In general, we need not consider the primary key update problem. However, there is an exception. If there is a keyword *"is_to_be_deleted"*, the SIC is relevant to any of the key attributes of the relationship R or entity E on update. The proper SIC name (refer to Appendix H.4) should be recorded in its *"associated_PKSIC_D"* predicate.

   (b) Suppose that the SIC also contains attributes of other than the relationship type R or entity type E; or contains assertions directly referencing R or E.

      i. If the SIC is relevant to the insertion or deletion of the relationship R, it would also be relevant to the update of any of its key attributes that relates to its sharing entities. The proper SIC name (see Appendix H.4) should be recorded in its *"associated_PKSIC_I"* or *"associated_PKSIC_D"* predicate (see Appendix B.2). It is not relevant to the update of the relationship's key attributes that relate to non-sharing entities.

      ii. If the SIC is relevant to the insertion or deletion of the entity E, it would be also relevant to the update of any of its primary key attributes. The proper SIC name (see Appendix H.4) should be recorded in its *"associated_PKSIC_I"* or *"associated_PKSIC_D"* predicate.

H.2 Write the Proper Precondition and Predicate Components

1. If there is any aggregate function with the attribute $E.A$ or $R.A$ as an argument, properly attach subscripts to the entity or relationship owning the attribute. That is, suppose *agg_fcn* is an aggregate function, do the following (similarly, for the $R.A$).

- change *"agg_fcn(E.A)"* to *"agg_fcn(E_1.A)"*;
- change *"agg_fcn(E.A) comp_op arithmetic_simple_expression involving E.A"*) to *"agg_fcn($\{E_1.A \mid E_1.A \neq E_0.A\}$) comp_op arithmetic_simple_expression involving E_0.A"*.

2. Suppose that the special predicate *unique* is used.
   - If it contains only one argument, e.g., *"unique(E.A)"*, change it to *"count($\{E_1 \mid E_1.A = E_0.A\}$)=1"*.
   - If it contains more than one argument, e.g., *"unique(E.A1, E.A2)"*, change it to *"count($\{E_1$ | comp_atts_occ(E_0, Comp_Key, CurrentCompAttsValue), comp_atts_occ(E_1, Comp_Key, AnyCompAttsValue), CurrentCompAttsValue=AnyCompAttsValue$\}$)=1"*, where *Comp_Key* (e.g.,{E.A1, E.A2}) is a set containing those arguments in the predicate *unique*.

3. If there is a relationship expression *"Entity_Occ1 Rship_Occ Entity_Occ2"*, e.g., *"E R F"*, rewrite it for sharing entities by using a special predicate *"rship_occ_part(Rship_Occ,Role_Type,Sharing_Entity_Occ)"* that is used to evaluate whether a *Sharing_Entity_Occ* participates in a relationship occurrence *Rship_Occ* with the *Role_Type*.
   - If there is no *"with_respect_to"* modifier, the *Role_Type* is the same as the entity type of *E*.
   - If there are *"with_respect_to"* modifiers with a common entity type, the *Role-Type* would be the entity type in these modifiers.
   - If those entity types in the *"with_respect_to"* modifiers are different, the relationship expressions would be written with the same entity variable, but different *Role_Types*. For example, supposing that we have *"if with_respect_to E, E RX F then with_respect_to G, G RY H"*, the first relationship expression would be written as *"rship_occ_part(RX, "E", E)"*, the second relationship expression would be written as *"rship_occ_part(RY, "G", E)"*,
   - If the entity types in *"with_respect_to"* modifiers are in ordered lists, there would be two pairs of *"rship_occ_part"* assertions with same entity variables taking their *Role_Types* in the corresponding order. For example, given *"if with_respect_to (E,F), E RX F then with_respect_to (H,G), G RY H"*, we would have the following four assertions:
     "rship_occ_part(RX, "E", E)",
     "rship_occ_part(RY, "F", F)",
     "rship_occ_part(RY, "H", E)", and
     "rship_occ_part(RY, "G", F)".

4. Omit any quantifier "$\forall$" or "$\exists$" of the SIC unless there is an *"at_least"*, *"at_most"*, *"exactly"*, or *"different"* following an "$\exists$" quantifier.

5. Rewrite numerical quantifiers. That is, for all numerical quantifiers on any relationship R, do the following.

   - change *"$\exists$ at_least Number R"* to *"count(R) $\geq$ Number"*;
   - change *"$\exists$ at_most Number R"* to *"count(R) $\leq$ Number"*;
   - change *"$\exists$ exactly Number R"* to *"count(R) = Number"*.

   A subscript may be attached to R in a later step.

6. If a *"before"* modifier is attached to an assertion referencing an object, apply an *old* function to that object and delete the *"before"* modifier.

7. Suppose that a general SIC is found to be relevant to an attribute (say $E.A$ or $R.A$) by applying the algorithm in Appendix H.1.

   (a) If the sub-SIC declares the attribute to be non-updateable, the predicate would only include *"false"*. All other assertions are in the precondition component to identify the attribute.

   (b) In other cases, the predicate component would usually only have the assertion (say Q) containing the attribute ($E.A$ or $R.A$) for which we are reformulating. That is,

      i. If the original general SIC contains only an arithmetic expression Q, leave it as the predicate.

      ii. Suppose that the original general SIC is in rule format.

         - If Q is in the *"if"* part, negate it and move it to the predicate component. Leave the further restrictions (if any) on Q in the precondition component. Negate all the original assertions in the *"then"* part of the general SIC, and move them to "AND" with the precondition component.
         - If Q is in the *"then"* part, leave it as the predicate component, move (but not negate) the further restrictions (if any) on Q to "AND" with the precondition component.

   (c) If the subscript 1 has been attached to its entity or relationship no matter whether it is attached to this attribute (e.g., *agg_fcn*($E_1.A$)) or other attributes (e.g., *agg_fcn*($E_1.B$)) of the same entity or relationship owing to the aggregate functions mentioned in step 1, attach the subscript 0 (i.e., $E_0.A$) to its assertions other than those involving aggregate functions.

(d) If the original general constraint is to assert explicitly the equality of some key attributes of two entity types, it should be dealt with specially. In the precondition component of the sub-SIC for the involved key attribute of one entity on update, the *old* function is used to search for the corresponding occurrence of the other entity before updating. Its predicate component should require the new values of the corresponding key attributes of the two entity types to be equivalent.

8. Suppose that a general SIC is found to be relevant to a relationship R or entity E by applying the algorithm in Appendix H.1.

   (a) If the SIC only contains assertions referencing to attributes of the relationship R or entity E, keep the original format that the system obtains from the database designer as the precondition and predicate components of the sub-SIC for the relationship R or entity E.

   (b) Suppose that the SIC also contains assertions referencing to attributes of other than the relationship R or entity E, or contains assertions directly on the relationship R or entity E.

      i. Suppose that the SIC is found to be relevant to the entity E on insertion. In general, the sub-SIC for the entity keeps the original format. However, if the assertion (say, Q) containing the entity is in the *"then"* part with a "not" or "¬", rewrite the original format — negate and move the Q to the precondition component, negate and move all other assertions in the original *"if"* part to the predicate component.
      Replace any *"count(E)"* with "$\exists\ E_1$, *count*(E_1)".

      ii. Suppose that the SIC is found to be relevant to the entity E on deletion. Rewrite the original format so that the assertion containing the entity E is in the precondition component.

      iii. Suppose that the SIC is found to be relevant to the relationship R on insertion. In general, keep the original format. However, if the *"if"* part originally does not contain those *"rship_occ_part"*s of the sharing entities regarding to the relationship R, add them to the precondition component to identify this specific relationship R. If a *"rship_occ_part"* assertion is originally in the *"then"* part with a *"not"* or "¬" attached to it, rewrite the original format so that the assertion is in the precondition component.
      Replace any *"count(R)"* with "$\exists\ R_1$, *count*(R_1)".

      iv. Suppose that the SIC is found to be relevant to the relationship R on deletion.

         A. Suppose that there is "$\exists$ *different* R" in the original format.
         Replace "$\exists$ *different* R" with "$\exists\ R_1$, $R_1 \neq R_0$".
         In its precondition component, replace the relationship variable R with R_0.

In its predicate component, replace the relationship variable R with R_1.

B. Suppose that it is not the above case A, and the assertion(s) *rship_occ_part* containing this relationship R is (are) in the *"if"* part.
Keep the original format after removing any *"previously"* modifier.
Replace any *"count(R)"* with "$\exists\ R_1,\ R_1 \neq R_0,\ count(R_1)$".

C. Suppose that it is not the above case A, and the assertion(s) (e.g., say Q) containing this relationship R is (are) in the *"then"* part.
   - Suppose that for maintaining semantic integrity it is possible to find other occurrences to replace the one to be deleted. That is, originally the relationship expression does not contain an "$\exists$ *at_most*" numerical quantifier; and the SIC is not relevant to the all key attributes of the relationship. (For example, a **Subset_Relationship SIC** is relevant to the whole relationship key.)
     - The original assertions in *"if"* part are in the precondition component.
     - In addition, add one copy of the assertion(s) Q containing this relationship R variable with a subscript 0 at the beginning of the precondition component to be in conjunction with other assertions.
     - If originally Q is a part of further restrictions on some assertions (say *Qsome*), move (but not negate) *Qsome*, after removing their numerical quantifiers or *count* aggregate functions, to the precondition component. Rearrange them in a reverse order and put them behind the assertion Q.
     - Replace the assertion(s) Q (that may include the *"count"* aggregate function) in the *"then"* part with the one containing this relationship variable with a subscript 1, and add the assertion "$\exists\ R_1,\ R_1 \neq R_0$".
     - The further restrictions (if any) on Q are kept unchanged in the predicate component.

     For example, suppose that Q_0 is the assertion(s) containing R_0, and Q_1 is the assertion(s) containing R_1. If originally we have *"if* QW *then* $Q \vee QX \vee QY \vee \ldots$ ", the precondition component would be "$Q_0 \wedge QW$", and the predicate component would be "$Q_1 \vee QX \vee QY \vee \ldots$ ". If originally we have *"if* QW *then* $QX \wedge QY \wedge Q \wedge QZ \wedge \ldots$ ", where QY contains a numerical quantifier "$\exists$ *at_least*", the precondition component would be "$Q_0 \wedge \widehat{QY} \wedge QX \wedge QW$", where $\widehat{QY}$ is the QY after removing the quantifier, and the predicate component would be "$Q_1 \wedge QZ \wedge \ldots$ ".
   - Suppose that it is not possible to find other occurrences to replace the one to be deleted.

- Move (but do not negate) Q and all its further restrictions to the precondition component.
- If after doing the above, we find that the predicate component contains only the assertion "*false*", negate all the assertions in original *"if"* part and move them to the predicate component.

v. Although in the above we have mentioned placing an "∃" quantifier on a variable with a subscript 1, an "∃" quantifier on a variable in other cases is optional. Without an explicit quantifier, an "∃" quantifier can be automatically assumed on any variable in that assertion. We may explicitly express the quantifier just for clarity. No explicit quantifier need be placed on the object for which we are writing precondition and predicate components. We may place an explicit "∃" quantifier on an unknown variable when it is its first time to be included in any assertion. For example, if we are writing precondition and predicate components for R and we need an assertion *rship_occ_part(R, "E", E)*, we may place an explicit quantifier on E, i.e., *"∃ E"*.

H.3 Suggest the Violation Action Component

1. A sub-SIC with certainty less than 100% (in a ratio scale), not 10 (supposing that the database designer specifies 10 as the highest in an ordinal scale), "uncertain" (in 2 levels), or fuzzy terms (e.g., "sometimes") would have at least two alternative violation actions: *"warning"* and *"conditionally_reject"*. However, if the database designer has specified a certainty threshold, those sub-SICs with certainty less than the threshold would only have one violation action — *"warning"*.

2. A sub-SIC with certainty 100% (in a ratio scale), 10 (if it is the highest in an ordinal scale), or "certain" (in 2 levels) would have at least one alternative violation action — *"reject"*.

3. Suppose that there are only two assertions on two objects (e.g., *"if (E RX F) then (E RY G)"*) in the SIC.

   (a) Suppose that these two assertions are both for relationships or entities (say *RX, RY*), not attributes. Both its sub-SICs would have propagation as an alternative violation action. That is, a certain sub-SIC could have an alternative — *"propagate"*; and an uncertain sub-SIC could have *"conditionally_propagate"*.

      i. In a sub-SIC for one object, the propagated operation type is opposite to the relevant operation of the other object found by the algorithm in Appendix H.1.
      For example, supposing that the general SIC is *"if (E RX F) then (E*

RY G)", the violation action in the sub-SIC for *RX* on insertion could be *"propagate(insert(RY))"*; supposing that the general SIC is *"if (E RX F) then ¬ (E RY G)"*, the violation action in the sub-SIC for *RX* on insertion could be *"propagate(delete(RY))"*.

ii. If there is an *"∃ at_least"* numerical quantifier attached to the assertion containing the object to be propagated or there is a *"count(RshipType) ≥ Number"* assertion in the sub-SIC, use the *"insert_all"* or *"delete_all"* as the propagated operation to indicate that there are explicit quantitative requirements when propagating to insert or delete the object occurrences.

iii. If the object to be propagated is an entity occurrence, specify into/from which entity type this occurrence is to be inserted or deleted. That is, we would need *"propagate(insert(EntType, E))"* or *"propagate(delete(EntType, E))"* where the *EntType* is the entity type with which *E* participates in a related relationship in the sub-SIC. This *EntType* is the one appeared in an assertion *"rship_occ_part(RshipType,EntType,E)"* or *"ent_occ(EntType,E)"* in the predicate or precondition component of the sub-SIC.

(b) Suppose that one of the assertions is for an attribute. Similarly, the sub-SIC for the attribute would have an alternative to propagate to insert or delete an entity or relationship. The other sub-SIC for the entity or relationship could have *"propagate(update(E.A, arithmetic_simple_expression))"* as an alternative only if the assertion of the attribute is *"E.A = arithmetic_simple_expression"*.

(c) Suppose that two assertions are both for attributes, say *E.A1* and *E.A2*. The sub-SIC for *E.A2* (or *E.A1*) could have an alternative: *"propagate(update(E.A1, arithmetic simple expression_1))"* (or *"propagate(update(E.A2, arithmetic simple expression_2))"*) only if those two assertions are
"E.A1 = arithmetic_simple_expression_1"
and *"E.A2 = arithmetic_simple_expression_2 "*;
or there is a single assertion such as *"E.A1 = E.A2"*.

4. In the case that there are two alternative violation actions in a sub-SIC, ask the database designer to choose one.

H.4 Generate the SIC Name

1. The *"ObjectType, OperationType"* in the SIC name can be decided by the system after applying the above algorithms.

2. The *"SICType"* and *"RelatedObjectTypeSet"* depend on the predefined SIC types (e.g., those in Appendix D). If there are some further restrictions on some predefined SIC types, in general, all the objects appeared in the further restrictions are included in the *"RelatedObjectTypeSet"*. However, the entity type names need

not be included if the names of their attributes have been included. Neither would the entity type names be included if a relationship type is mentioned and both of its participant entity types are concerned (e.g., **Subset_Relationship SIC**). No duplicate object names are included.

3. For some complicated applications, if the system finds that it cannot distinguish two SIC names by the previous steps, a *"SequenceNo"* is added.

Appendix I

Some Examples of SIC Reformulation and Decomposition

This appendix contains four examples of reformulating and decomposing general SICs.

Conditional_Value SIC:
Suppose that there are only two attributes involved, such as,
if E.A1 comp_op1 v1
then E.A2 comp_op2 v2
where *comp_op1* or *comp_op2* denotes $=, \leq, \geq, <$, or $>$.
This type of SIC is represented by the following three sub-SICs.

E.A2-U-ConditionalValue-(E.A1)

CERTAINTY	certain
FOR	E.A2
ON	update
IF	E.A1 comp_op1 v1
ASSERT	E.A2 comp_op2 v2
ELSE	reject

E.A1-U-ConditionalValue-(E.A2)

CERTAINTY	certain
FOR	E.A1
ON	update
IF	E.A2 $\neg$ comp_op2 v2
ASSERT	E.A1 $\neg$ comp_op1 v1
ELSE	reject

E-I-ConditionalValue-(E.A1, E.A2)

CERTAINTY	certain
FOR	E
ON	insertion
IF	E.A1 comp_op1 v1
ASSERT	E.A2 comp_op2 v2
ELSE	reject

Relationship_Depends_on_Relationship — involving 3 entity types:
For example, in Figure 5.2 (on page 110):
if E RX F
then E RY G
There are two decomposed sub-SICs to represent this type of SIC.

RX-I-RshipDepRship3E-(E,RY)

CERTAINTY	certain
FOR	RX
ON	insertion
IF	∃ E, rship_occ_part(RX,"E",E)
ASSERT	∃ RY, rship_occ_part(RY,"E",E)
ELSE	reject or propagate(insert(RY))

RY-D-RshipDepRship3E-(E,RX)

CERTAINTY	certain
FOR	RY
ON	deletion
IF	∃ E, rship_occ_part(RY_0,"E",E),
	∃ RX, rship_occ_part(RX,"E",E)
ASSERT	∃ RY_1, $RY_1 \neq RY_0$,
	rship_occ_part(RY_1,"E",E)
ELSE	reject or propagate(delete(RX))

Relationship-Depends-on-Relationship — involving 4 entity types:
In Figure 5.3, we may have
if E RX F
then (E RY G) ∨ (E RZ H)

RX-I-RshipDepRship4E-(E,RY,RZ)

CERTAINTY	certain
FOR	RX
ON	insertion
IF	∃ E, rship_occ_part(RX,"E",E)
ASSERT	(∃ RY, rship_occ_part(RY,"E",E)) ∨
	(∃ RZ, rship_occ_part(RZ,"E",E))
ELSE	reject

RY-D-RshipDepRship4E-(E,RX,RZ)

CERTAINTY	certain
FOR	RY
ON	deletion
IF	∃ E, rship_occ_part(RY_0,"E",E),
	∃ RX, rship_occ_part(RX,"E",E)
ASSERT	(∃ RY_1, $RY_1 \neq RY_0$, rship_occ_part(RY_1,"E",E)) ∨
	(∃ RZ, rship_occ_part(RZ,"E",E))
ELSE	reject

RZ-D-RshipDepRship4E-(E,RX,RY)

CERTAINTY	certain
FOR	RZ
ON	deletion
IF	∃ E, rship_occ_part(RZ_0,"E",E),
	∃ RX, rship_occ_part(RX,"E",E)
ASSERT	(∃ RY, rship_occ_part(RY,"E",E)) ∨
	(∃ RZ_1, $RZ_1 \neq RZ_0$, rship_occ_part(RZ_1,"E",E))
ELSE	reject

Subtype_Stronger_Totality_Cardinality:
For example, in the context of Figure 5.2, we may have:
if ∃ *RY, E RY G*
then ∃ *at_least Sub_Cmin RX, E RX F,*
∃ *RZ, F RZ H*

RY-I-SubStrongTotal-(E,F,RZ,RX)

CERTAINTY	certain
FOR	RY
ON	insertion
IF	∃ E, rship_occ_part(RY,"E",E)
ASSERT	∃ RX, rship_occ_part(RX,"E",E), ∃ F, rship_occ_part(RX,"F",F), ∃ RZ, rship_occ_part(RZ,"F",F), count(RX) ≥ Sub_Cmin
ELSE	reject

RX-D-SubStrongTotal-(E,F,RY,RZ)

CERTAINTY	certain
FOR	RX
ON	deletion
IF	∃ E, rship_occ_part(RX_0,"E",E), ∃ RY, rship_occ_part(RY,"E",E)
ASSERT	∃ RX_1, $RX_1 \neq RX_0$, rship_occ_part(RX_1,"E",E), ∃ F, rship_occ_part(RX_1,"F",F), ∃ RZ, rship_occ_part(RZ,"F",F), count(RX_1) ≥ Sub_Cmin
ELSE	reject

RZ-D-SubStrongTotal-(E,F,RY,RX)

CERTAINTY	certain
FOR	RZ
ON	deletion
IF	$\exists$ F, rship_occ_part(RZ_0,"F",F),
	$\exists$ RX, rship_occ_part(RX,"F",F),
	$\exists$ E, rship_occ_part(RX,"E",E),
	$\exists$ RY, rship_occ_part(RY,"E",E)
ASSERT	$\exists$ RZ_1, $RZ_1 \neq RZ_0$,
	rship_occ_part(RZ_1,"F",F)
ELSE	reject

Appendix J

Generic SIC Representation in the E-R-SIC Model

This appendix contains a partial listing of generic SIC representation in the E-R-SIC model for illustration. If there are two alternative violation actions, the system will query the database designer to choose one. Please refer to [Yang, 1992] for other generic SICs mentioned in Section 7.3.1.

Domain Constraint:

Entity-I-Domain*

CERTAINTY	certain
FOR	Entity*
ON	insertion
IF	entity("Entity*", Prim_Key, Comp_Key_Set, Ab_Max_Card)
ASSERT	set{AttName |
	attribute("Entity*", AttName, Domain,
	SpecialVRange, Null?, Unique?, Key?, Changeable?)},
	att_occ(Entity*, AttName, EntAttOcc),
	concatenate_SICname("Entity*.", AttName,
	"U-DomNull", AttDomainSIC1),
	checkcomSIC(AttDomainSIC1, EntAttOcc),
	concatenate_SICname("Entity*.", AttName,
	"U-DomTypeForVal", AttDomainSIC2),
	checkcomSIC(AttDomainSIC2, EntAttOcc),
	concatenate_SICname("Entity*.", AttName,
	"U-DomUnique", AttDomainSIC3),
	checkcomSIC(AttDomainSIC3, EntAttOcc)
ELSE	reject

Entity.Attribute*-U-DomNull*

CERTAINTY	certain
FOR	Entity*.Attribute*
ON	update
IF	attribute("Entity*", "Attribute*", Domain, SpecialVRange, Null?, Unique?, Key?, Changeable?), Null?=no
ASSERT	is_not_null(Entity*.Attribute*)
ELSE	reject

Entity.Attribute*-U-DomTypeFormatVal*

CERTAINTY	certain
FOR	Entity*.Attribute*
ON	update
IF	is_not_null(Entity*.Attribute*), attribute("Entity*", "Attribute*", Domain, SpecialVRange, Null?, Unique?, Key?, Changeable?), domain(Domain, DataType, Format, ValueRange)
ASSERT	satisfy_datatype(Entity*.Attribute*, DataType), satisfy_format(Entity*.Attribute*, Format), satisfy_value(Entity*.Attribute*, SpecialVRange), satisfy_value(Entity*.Attribute*, ValueRange)
ELSE	reject

Entity.Attribute*-U-DomUnique*

CERTAINTY	certain
FOR	Entity*.Attribute*
ON	update
IF	attribute("Entity*", "Attribute*", Domain, SpecialVRange, Null?, Unique?, Key?, Changeable?), Unique?=yes
ASSERT	count(set{Entity_1^* | Entity_0^*.Attribute* = Entity_1^*.Attribute*})= 1
ELSE	reject

Entity.Attribute*-U-DomChange*

CERTAINTY	certain
FOR	Entity*.Attribute*
ON	update
IF	attribute("Entity*", "Attribute*", Domain, SpecialVRange, Null?, Unique?, Key?, Changeable?), Changeable?=no
ASSERT	false
ELSE	reject

Primary_Key SIC:

Entity.Attribute*-U-PrimKeySurroDel*

CERTAINTY	certain
FOR	Entity*.Attribute*
ON	update
IF	entity("Entity*", Prim_Key, Comp_Key_Set, Ab_Max_Card), associated_PKSICs_D("Entity*", SIC_Name_Set), belongs_to("Attribute*", Prim_Key)
ASSERT	set{SIC_Name | belongs_to(SIC_Name, SIC_Name_Set)}, checkmemSIC(SIC_Name, old(Entity*))
ELSE	reject

Entity.Attribute*-U-PrimKeySurroIns*

CERTAINTY	certain
FOR	Entity*.Attribute*
ON	update
IF	entity("Entity*", Prim_Key, Comp_Key_Set, Ab_Max_Card), associated_PKSICs_I("Entity*", SIC_Name_Set), belongs_to("Attribute*", Prim_Key)
ASSERT	set{SIC_Name | belongs_to(SIC_Name, SIC_Name_Set)}, checkmemSIC(SIC_Name, new(Entity*))
ELSE	reject

Absolute Maximum Cardinality Constraint of an Entity Type:

Entity-I-AbsCard*

CERTAINTY	certain
FOR	Entity*
ON	insertion
IF	entity("Entity*", Primary_Key, Composite_Key_Set, Abs_Max_Card), Abs_Max_Card $\neq$ "*"
ASSERT	$\exists$ Entity_1^*, $\text{count}(\text{Entity}_1^*) \leq$ Abs_Max_Card
ELSE	reject

Incidence Constraint:

Relationship-I-Incidence-(EntType)*

CERTAINTY	certain
FOR	Relationship*
ON	insertion
IF	relationship_participant("Relationship*", EntType, Min_Cardinality, Max_Cardinality)
ASSERT	∃ EntOcc, ent_occ(EntType, EntOcc), rship_occ_part(Relationship*, EntType, EntOcc)
ELSE	reject or propagate(insert(EntType,EntOcc))

Entity-D-Incidence-(RshipType)*

CERTAINTY	certain
FOR	Entity*
ON	deletion
IF	relationship_participant(RshipType, "Entity*", Min_Cardinality, Max_Cardinality)
ASSERT	¬ (∃ RshipOcc, rship_occ(RshipType, RshipOcc), rship_occ_part(RshipOcc, "Entity*", Entity*))
ELSE	reject or propagate(delete(RshipOcc))

Subset_Relationship SIC: There are two decomposed sub-SICs to represent this type of SIC as below.

Relationship-I-RshipSubset-(AnotherRshipType)*

CERTAINTY	certain
FOR	Relationship*
ON	insertion
IF	subset_rship(("Relationship*",RoleType1,RoleType2), (AnotherRshipType,RoleType3,RoleType4)), rship_occ_part(Relationship*, RoleType1,EntOcc1), rship_occ_part(Relationship*, RoleType2, EntOcc2)
ASSERT	∃ RshipOcc, rship_occ(AnotherRshipType,RshipOcc), rship_occ_part(RshipOcc, RoleType3,EntOcc1), rship_occ_part(RshipOcc, RoleType4,EntOcc2)
ELSE	reject or propagate(insert(RshipOcc))

Relationship-D-RshipSubset-(AnotherRshipType)*

CERTAINTY	certain
FOR	Relationship*
ON	deletion
IF	subset_rship((AnotherRshipType,RoleType3,RoleType4), ("Relationship*",RoleType1,RoleType2)), rship_occ_part(Relationship*, RoleType1, EntOcc1), rship_occ_part(Relationship*, RoleType2, EntOcc2)
ASSERT	¬ (∃ RshipOcc, rship_occ(AnotherRshipType,RshipOcc), rship_occ_part(RshipOcc, RoleType3, EntOcc1), rship_occ_part(RshipOcc, RoleType4, EntOcc2))
ELSE	reject or propagate(delete(RshipOcc))

Relationships_Union Special SIC:
The following sub-SICs assert that *if SuperRship then (SubRship1 ∨ SubRship2 ...).*

Relationship-I-RshipUnion-(SubRships)*

CERTAINTY	certain
FOR	Relationship*
ON	insertion
IF	rships_union_condition(("Relationship*",RoleType1,RoleType2), SubRships), rship_occ_part(Relationship*, RoleType1, EntOcc1), rship_occ_part(Relationship*, RoleType2, EntOcc2)
ASSERT	∃ SubRshipOcc, rship_occ(SubRship,SubRshipOcc), belongs_to((SubRship,RoleType3,RoleType4), SubRships), rship_occ_part(SubRshipOcc, RoleType3,EntOcc1), rship_occ_part(SubRshipOcc, RoleType4,EntOcc2)
ELSE	reject

Relationship-D-RshipUnion-(SuperRship, OtherSubRships)*

CERTAINTY	certain
FOR	Relationship*
ON	deletion
IF	rships_union_condition((SuperRship,RoleType1,RoleType2), SubRships), belongs_to(("Relationship*",RoleTypei,RoleTypej), SubRships), rship_occ_part(Relationship*, RoleTypei, EntOcc1), rship_occ_part(Relationship*, RoleTypej, EntOcc2), ∃ SuperRshipOcc, rship_occ(SuperRship,SuperRshipOcc), rship_occ_part(SuperRshipOcc, RoleType1,EntOcc1), rship_occ_part(SuperRshipOcc, RoleType2,EntOcc2)
ASSERT	remove_from_set(SubRships, ("Relationship*",RoleTypei,RoleTypej), OtherSubRships), ∃ AnotherRshipOcc, rship_occ(AnotherSubRship, AnotherRshipOcc), belongs_to((AnotherSubRship,RoleTypek,RoleTypel), OtherSubRships), rship_occ_part(AnotherRshipOcc, RoleTypek,EntOcc1), rship_occ_part(AnotherRshipOcc, RoleTypel, EntOcc2)
ELSE	reject

Exclusive_Relationship SIC:

Relationship-I-RshipExclusive-(SharingEnt1, SharingEnt2, AnotherRshipType)*

CERTAINTY	certain
FOR	Relationship*
ON	insertion
IF	ex_rships(ExRshipSet), belongs_to(("Relationship*",SharingEnt1), ExRshipSet), $\exists$ AnotherRshipType, AnotherRshipType $\neq$ "Relationship*", belongs_to((AnotherRshipType,SharingEnt2), ExRshipSet), rship_occ_part(Relationship*, SharingEnt1,EntOcc)
ASSERT	$\neg$ ($\exists$ RshipOcc, rship_occ(AnotherRshipType,RshipOcc), rship_occ_part(RshipOcc, SharingEnt2,EntOcc))
ELSE	reject or propagate(delete(RshipOcc))

Totality Constraints:

Entity-I-Totality-(RshipType)*

CERTAINTY	certain
FOR	Entity*
ON	insertion
IF	relationship_participant(RshipType, "Entity*", Cmin, Cmax), Cmin $\geq$ 1
ASSERT	$\exists$ RshipOcc, rship_occ(RshipType,RshipOcc), rship_occ_part(RshipOcc, "Entity*", Entity*), count(RshipOcc) $\geq$ Cmin
ELSE	reject or propagate(insert_all(RshipOcc))

Relationship-D-Totality-(EntType)*

CERTAINTY	certain
FOR	Relationship*
ON	deletion
IF	relationship_participant("Relationship*", EntType, Cmin, Cmax), Cmin $\geq$ 1, rship_occ_part(Relationship^*_0, EntType, EntOcc)
ASSERT	$\exists$ Relationship^*_1, $\text{Relationship}^*_1 \neq \text{Relationship}^*_0$, rship_occ_part($\text{Relationship}^*_1$, EntType, EntOcc), count(Relationship^*_1) $\geq$ Cmin
ELSE	reject or propagate(delete(EntType,EntOcc))

Relative Maximum Cardinality Constraint:

Relationship-I-RshipMaxCard-(EntType)*

CERTAINTY	certain
FOR	Relationship*
ON	insertion
IF	relationship_participant("Relationship*", EntType, Cmin, Cmax), Cmax $\neq$ "*", ent_occ(EntType, EntOcc), rship_occ_part(Relationship$^*_0$, EntType, EntOcc)
ASSERT	$\exists$ Relationship$^*_1$, rship_occ_part(Relationship$^*_1$, EntType, EntOcc), count(Relationship$^*_1$) $\leq$ Cmax
ELSE	reject

Appendix K

Algorithms Transforming SICs to Relational Form

This appendix proposes algorithms to transform SIC representations in the E-R-SIC model into corresponding ones in the relational model.

K.1 Transform the SIC Representation

1. Examine the relationships mentioned in a sub-SIC in the E-R-SIC model. If all the relationships mentioned in that sub-SIC are represented by separate relations, do nothing — its representation in the relational model is the same as in the E-R-SIC model.

2. Suppose that the sub-SIC is <u>not</u> a **Primary_Key SIC**. For each relationship type (say R) that is represented by a foreign key in an entity relation type (say H), do the following.

   (a) Transform the components of the precondition and predicate.

      i. In general, do the following.

         - Replace all assertions in which R appears, e.g., *rship_occ_part(R, RoleType,E)*, where E is any entity occurrence variable, with *foreign_ent_occ("R", H, E)*. If R has a subscript (either 0 or 1) in some assertion, H should be attached with the same subscript in that assertion. If an *old* function is applied to R in the *rship_occ_part* assertion, apply the *old* function to H in the *foreign_ent_occ* assertion.
         - Suppose that a subscript is attached to H according to the above step and H is also a participant in another relationship (say RA).
           - If its *rship_occ_part(RA, RoleType,H)* assertion is in the precondition component, attach a subscript 0 to H in that assertion.
           - If its *rship_occ_part(RA, RoleType,H)* assertion is in the predicate component, attach a subscript 1 to H in that assertion.
         - Replace "*count(R)*" with "*count(H)*" (use the same subscript if any).
         - Replace any quantifier on R with the same quantifier on H (and use the same subscript if any).

ii. However, suppose that there is an assertion requiring an entity occurrence of the other entity type (say G) to participate in some occurrence (say, $\hat{R}$) of the same relationship type R with the entity type H. That is, there is an assertion such as *rship_occ_part($\hat{R}$, "H", G)*.
For such kind of $\hat{R}$, do the following instead of (i).

- Replace all assertions in which $\hat{R}$ appears, e.g., *rship_occ_part($\hat{R}$, RoleType,F)*, where F is any entity occurrence variable, with *foreign_ent_occ("R", G, F)*. If $\hat{R}$ has a subscript (either 0 or 1) in some assertion, G should be attached with the same subscript in that assertion. If an *old* function is applied to $\hat{R}$ in the *rship_occ_part* assertion, apply the *old* function to G in the *foreign_ent_occ* assertion.
- Suppose that a subscript is attached to G according to the above step and G is also a participant in another relationship (say RB).
  - If its *rship_occ_part(RB, RoleType,G)* assertion is in the precondition component, attach a subscript 0 to G in that assertion.
  - If its *rship_occ_part(RB, RoleType,G)* assertion is in the predicate component, attach a subscript 1 to G in that assertion.
- Replace "*count($\hat{R}$)*" with "*count(G)*" (use the same subscript if any).
- Replace any quantifier on R with the same quantifier on H (and use the same subscript if any).

(b) In the object component, if the object happens to the relationship R, replace R with H; otherwise, do nothing.

(c) In the violation action component, if the action is not to propagate some operations of R, do nothing; otherwise, do the following.

- Replace the argument *(R)* of the propagated operation (e.g., *delete*) with the pair, *("H",H)* (e.g., *propagate(delete("H",H))*.
- Replace the argument *($\hat{R}$)* of the propagated operation (e.g., *delete*) with the pair, *("H",G)* (e.g., *propagate(delete("H",G))*.

(d) Transform the SIC name.

i. If R is the first part (i.e., R is the object to be checked), replace it with H. In addition, add R to the *RelatedObjectTypeSet*. If originally H is in the *RelatedObjectTypeSet*, remove it from that set.

ii. If R is not the first part, it must already be in the *RelatedObjectTypeSet*. Keep it and also add H to it.

(e) Examine the transformed sub-SIC. Delete any *foreign_ent_occ* predicate that has the equivalent second and third arguments (e.g., *"foreign_ent_occ("R", H, H)"* or *"foreign_ent_occ("R", G, G)"*), that is, its foreign entity relation is the same as the entity relation that the relationship R is hidden. Also, if there are duplicate quantifiers, only keep the first one.

(f) Suppose that the primary keys of E and F are *E.Ekey* and *F.Fkey*, respectively. After doing the above, check the transformed sub-SIC.

- If any of the following occurs, the transformed sub-SIC is redundant, so remove it.
  - Its predicate component becomes empty.
  - Its predicate component contains a single assertion *count(H_1) comp_op Number*, or *count(G_1) comp_op Number*, where *comp_op* is a "=", "≤" or "≥", and *Number* is a positive integer number (e.g., 1).
  - Its precondition component has an assertion *foreign_ent_occ("R", H, E)*, and its predicate component contains only *H.Ekey=E.Ekey*.
  - Its precondition component has an assertion foreign_ent_occ("R", G, F), and its predicate component contains only *G.Fkey=F.Fkey*.
- If the predicate component contains a *count(H)* ≥ *1* assertion (no subscript other than 0 is attached to *H*), change it to "is_not_null(H.Ekey)".
- If the predicate component contains a *count(G)* ≥ *1* assertion (no subscript other than 0 is attached to *G*), change it to "is_not_null(G.Fkey)".

3. Suppose that the sub-SIC is a **Primary_Key SIC** and the involved relationship R is represented by adding the primary key of an entity type E to another entity type H as a foreign key. Also, suppose that the primary keys of E and H are *E.Ekey* and *H.Hkey*, respectively. If the sub-SIC is for *R.Hkey* and we already have a **Primary_Key SIC** representation for *H.Hkey*, delete this sub-SIC representation (because we will merge the related *SIC_Name_Sets* according to the algorithm in Appendix K.2). Otherwise, perform the following transformation.

   (a) In the precondition and predicate components, replace "*old(R)*" and "*new(R)*" with "*old(H)*" and "*new(H)*", respectively.

      i. If the sub-SIC is for *R.Ekey*, replace *associated_PKSICs_I("R", "Ekey", SIC_Name_Set1)* and *associated_PKSICs_D("R", "Ekey", SIC_Name_Set2)* with *associated_FKSICs_I("H", "Ekey", SIC_Name_Set1)* and *associated_FKSICs_D("H", "Ekey", SIC_Name_Set2)*, respectively.

      ii. If the sub-SIC is for *R.Hkey*, replace *associated_PKSICs_I("R", "Hkey", SIC_Name_Set3)* and *associated_PKSICs_D("R", "Hkey", SIC_Name_Set4)* with *associated_PKSICs_I("H", SIC_Name_Set3)* and *associated_PKSICs_D("H", SIC_Name_Set4)*, respectively.

   (b) In the object component, change "*R.Hkey*" (or "*R.Ekey*") to "*H.Hkey*" (or "*H.Ekey*").

   (c) Change the SIC name by replacing "*R.Hkey*" (or "*R.Ekey*") with "*H.Hkey*" (or "*H.Ekey*") in its first part and adding R to its *RelatedObjectTypeSet*.

K.2 Construct SIC Name Sets for the Foreign Key Update

For each relationship type (say R) that is represented by adding the key of one entity type (say F) to another entity relation type (say H) as a foreign key, do the following.

1. Put the SIC names (if any) in the original *SIC_Name_Set* of the *associated_PKSICs_I* predicate of R's key attributes, which are related to F's key attributes, into the *SIC_Name_Set* of the *associated_FKSICs_I* predicate (see Appendix B.2) of H after properly changing the SIC names.

2. Similarly, put the SIC names (if any) in the original *SIC_Name_Set* of the *associated_PKSICs_D* predicate of R's key attributes, which are related to F's key attributes, into the *SIC_Name_Set* of the *associated_FKSICs_D* predicate of H after properly changing the SIC names.

3. Merge the SIC names (if any) in the original *SIC_Name_Set* of the *associated_PKSICs_I* predicate of R's key attributes, which are related to the key attributes of H, with the SIC names in the *associated_PKSICs_I* predicate of H after properly changing the SIC names.

4. Similarly, merge the SIC names (if any) in the original *SIC_Name_Set* of the *associated_PKSICs_D* predicate of R's key attributes, which are related to the key attributes of H, with the SIC names in the *associated_PKSICs_D* predicate of H after properly changing the SIC names.

Appendix L

Some Examples of Transforming SICs to Relational Form

This appendix contains three examples of transforming the SICs of Appendix I in the E-R-SIC model into corresponding ones in the relational model.

Relationship_Depends_on_Relationship — Involving 3 entity types:
For example, in the context of Figure 5.2 (on page 110):
if E RX F
then E RY G
The representation of sub-SICs in the E-R-SIC model is shown on page 250.

There are four possible sets of representations in the relational model:
Representation 1: Both *RX* and *RY* are represented by separate relations. The two sub-SICs are exactly the same as those in the E-R-SIC model.

Representation 2: *RX* is hidden in the entity relation *F*, but *RY* is represented by a separate relation.

F-I-RshipDepRship3E-(E,RY,RX)

CERTAINTY	certain
FOR	F
ON	insertion
IF	∃ E, foreign_ent_occ("RX",F,E)
ASSERT	∃ RY, rship_occ_part(RY,"E",E)
ELSE	reject or propagate(insert(RY))

RY-D-RshipDepRship3E-(E,RX,F)

CERTAINTY	certain
FOR	RY
ON	deletion
IF	$\exists$ E, rship_occ_part(RY_0,"E",E),
	$\exists$ F, foreign_ent_occ("RX",F,E)
ASSERT	$\exists$ RY_1, $RY_1 \neq RY_0$,
	rship_occ_part(RY_1,"E",E)
ELSE	reject or propagate(delete("F",F))

Representation 3: *RY* is hidden in the entity relation *G*, but *RX* is represented by a separate relation.

RX-I-RshipDepRship3E-(E,RY,G)

CERTAINTY	certain
FOR	RX
ON	insertion
IF	$\exists$ E, rship_occ_part(RX,"E",E)
ASSERT	$\exists$ G, foreign_ent_occ("RY",G,E)
ELSE	reject or propagate(insert("G",G))

G-D-RshipDepRship3E-(E,RX,RY)

CERTAINTY	certain
FOR	G
ON	deletion
IF	$\exists$ E, foreign_ent_occ("RY",G_0,E),
	$\exists$ RX, rship_occ_part(RX,"E",E)
ASSERT	$\exists$ G_1, $G_1 \neq G_0$,
	foreign_ent_occ("RY",G_1,E)
ELSE	reject or propagate(delete(RX))

Representation 4: *RX* is hidden in the entity relation *F*, and *RY* is hidden in the entity relation *G*.

F-I-RshipDepRship3E-(E,RY,RX,G)

CERTAINTY	certain
FOR	F
ON	insertion
IF	∃ E, foreign_ent_occ("RX",F,E),
ASSERT	∃ G, foreign_ent_occ("RY",G,E)
ELSE	reject or propagate(insert("G",G))

G-D-RshipDepRship3E-(E,RX,RY,F)

CERTAINTY	certain
FOR	G
ON	deletion
IF	∃ E, foreign_ent_occ("RY",G_0,E),
	∃ F, foreign_ent_occ("RX",F,E)
ASSERT	∃ G_1, $G_1 \neq G_0$,
	foreign_ent_occ("RY",G_1,E)
ELSE	reject or propagate(delete("F",F))

Relationship_Depends_on_Relationship — Involving 4 entity types:
In the Figure 5.3, we may have
if E RX F
then (E RY G) ∨ (E RZ H)
The representation of sub-SICs in the E-R-SIC model is shown on page 251. Since, if such a SIC exists, *E* cannot have (1,1) cardinalities in any of the above relationship types, there are eight possible combinations of representations. Only one is listed here, where all relationship types are represented by foreign keys.

F-I-RshipDepRship4E-(E,RY,RZ,RX,G,H)

CERTAINTY	certain
FOR	*F*
ON	insertion
IF	∃ E, foreign_ent_occ("RX",F,E)
ASSERT	(∃ G, foreign_ent_occ("RY",G,E)) ∨
	(∃ H, foreign_ent_occ("RZ",H,E))
ELSE	reject

G-D-RshipDepRship4E-(E,RX,RZ,RY,F,H)

CERTAINTY	certain
FOR	G
ON	deletion
IF	∃ E, foreign_ent_occ("RY",G_0,E),
	∃ F, foreign_ent_occ("RX",F,E)
ASSERT	(∃ G_1, $G_1 \neq G_0$, foreign_ent_occ("RY",G_1,E)) ∨
	(∃ H, foreign_ent_occ("RZ",H,E))
ELSE	reject

H-D-RshipDepRship4E-(E,RX,RY,RZ,F,G)

CERTAINTY	certain
FOR	H
ON	deletion
IF	∃ E, foreign_ent_occ("RZ",H_0,E),
	∃ F, foreign_ent_occ("RX",F,E)
ASSERT	(∃ G, foreign_ent_occ("RY",G,E)) ∨
	(∃ H_1, $H_1 \neq H_0$, foreign_ent_occ("RZ",H_1,E))
ELSE	reject

Subtype_Stronger_Totality Constraint:
For example, in the context of Figure 5.2, we may have:
if ∃ *RY, E RY G*
then ∃ *at_least Sub_Cmin RX, E RX F,*
∃ *RZ, F RZ H*

The representation of sub-SICs in the E-R-SIC model is shown on page 252. Here only one possible representation in the relational model is listed. Suppose that in the relational model, *RY* is represented by a foreign key of *G*, *RX* is represented by a foreign key of *F*, *RZ* is represented by a foreign key of *H*. (It would be not a subtype_stronger_totality constraint if *F* has (1,1) cardinalities in *RZ*.)

G-I-SubStrongTotal-(E,F,RZ,RX,RY,H)

CERTAINTY	certain
FOR	*G*
ON	insertion
IF	∃ E, foreign_ent_occ("RY",G,E)
ASSERT	∃ F, foreign_ent_occ("RX",F,E), ∃ H, foreign_ent_occ("RZ",H,F), count(F) ≥ Sub_Cmin
ELSE	reject

F-D-SubStrongTotal-(E,RY,RZ,RX,G,H)

CERTAINTY	certain
FOR	F
ON	deletion
IF	∃ E, foreign_ent_occ("RX",F_0,E), ∃ G, foreign_ent_occ("RY",G,E)
ASSERT	∃ F_1, $F_1 \neq F_0$, foreign_ent_occ("RX",F_1,E), ∃ H, foreign_ent_occ("RZ",H,F_1), count(F_1) ≥ Sub_Cmin
ELSE	reject

H-D-SubStrongTotal-(E,F,RY,RX,RZ)

CERTAINTY	certain
FOR	H
ON	deletion
IF	$\exists$ F, foreign_ent_occ("RZ",H_0,F), $\exists$ E, foreign_ent_occ("RX",F,E) $\exists$ G, foreign_ent_occ("RY",G,E)
ASSERT	$\exists$ H_1, $H_1 \neq H_0$, foreign_ent_occ("RZ",H_1,F)
ELSE	reject

Appendix M

Generic SIC Representation in the Relational Model

This appendix contains a partial listing of SIC representations for generic *Relationship**, *Entity**, Relationship*.Attribute*, and *Entity*.Attribute** in the relational model for illustration. If there are two alternative violation actions, the system will query the database designer to choose one. Please refer to [Yang, 1992] for more examples of generic SICs in the relational model.

Incidence Constraint:

Entity-I-Incidence-(ForeignEntType)*

CERTAINTY	certain
FOR	Entity*
ON	insertion
IF	relationship_hidden_entity(RshipType, "Entity*", ForeignEntType, Cmin,Cmax)
ASSERT	foreign_ent_occ(RshipType, Entity*, FEntOcc), ∃ FEntOcc, ent_occ(ForeignEntType, FEntOcc)
ELSE	reject or propagate(insert(ForeignEntType,FEntOcc))

Entity-D-Incidence-(RshipType,EntType)*

CERTAINTY	certain
FOR	Entity*
ON	deletion
IF	relationship_hidden_entity(RshipType, EntType, "Entity*",Cmin,Cmax)
ASSERT	¬ (∃ EntOcc, ent_occ(EntType, EntOcc), foreign_ent_occ(RshipType, EntOcc,Entity*))
ELSE	reject or propagate(delete(EntType,EntOcc))

Subset_Relationship SIC:
In this SIC type, it is impossible that an entity type has cardinalities (1,1) in both relationship types (nor could each of the corresponding entity types in a specialization hierarchy have (1,1) in its relationship type). There are four sets of representations in the relational model for this type.
Representation 1: Both relationships are represented by separate relations. In this case, the sub-SICs are exactly the same as those in the E-R-SIC model.

Representation 2: The subset relationship is represented by a foreign key, the super-set is not. It covers two possible combinations. (For example, if R is the subset relationship type between entity types E and F, it can be represented by a foreign key either in E or in F.)

Entity-I-RshipSubset-(RshipType,AnotherRshipType)*

CERTAINTY	certain
FOR	Entity*
ON	insertion
IF	relationship_hidden_entity(RshipType, "Entity*", EntType2, Cmin1,Cmax1), subset_rship((RshipType,RoleType1,RoleType2), (AnotherRshipType,RoleType3,RoleType4)), relationship_participant(AnotherRshipType, RoleType3, Cmin3, Cmax3), ∃ EntOcc2, foreign_ent_occ(RshipType,Entity*,EntOcc2), which_foreign("Entity*", RoleType1, RoleType2, EntOcci, EntOccj, Entity*, EntOcc2)
ASSERT	∃ RshipOcc, rship_occ(AnotherRshipType,RshipOcc), rship_occ_part(RshipOcc, RoleType3, EntOcci), rship_occ_part(RshipOcc, RoleType4, EntOccj)
ELSE	reject or propagate(insert(RshipOcc))

Relationship-D-RshipSubset-(AnotherRshipType,EntType1)*

CERTAINTY	certain
FOR	Relationship*
ON	deletion
IF	subset_rship((AnotherRshipType,RoleType1,RoleType2), ("Relationship*",RoleType3,RoleType4)), relationship_hidden_entity(AnotherRshipType, EntType1, EntType2, Cmin1,Cmax1), relationship_participant("Relationship*", RoleType3, Cmin3, Cmax3), rship_occ_part(Relationship*, RoleType3, EntOcc3), rship_occ_part(Relationship*, RoleType4, EntOcc4), which_foreign(EntType1, RoleType1, RoleType2, EntOcc3, EntOcc4, EntOcci, FEntOcci)
ASSERT	¬ foreign_ent_occ(AnotherRshipType,EntOcci,FEntOcci)
ELSE	reject or propagate(delete(EntType1,EntOcci))

Representation 3: The super-set relationship is represented by a foreign key, the subset is not. It covers two possible combinations.

Relationship-I-RshipSubset-(AnotherRshipType)*

CERTAINTY	certain
FOR	Relationship*
ON	insertion
IF	subset_rship(("Relationship*",RoleType1,RoleType2), (AnotherRshipType,RoleType3,RoleType4)), relationship_participant("Relationship*", RoleType1, Cmin1, Cmax1), relationship_hidden_entity(AnotherRshipType, EntType3, EntType4, Cmin4,Cmax4), rship_occ_part(Relationship*, RoleType1,EntOcc1), rship_occ_part(Relationship*, RoleType2, EntOcc2), which_foreign(EntType3, RoleType3, RoleType4, EntOcc1, EntOcc2, EntOcci, FEntOcci)
ASSERT	foreign_ent_occ(AnotherRshipType,EntOcci,FEntOcci)
ELSE	reject or propagate(insert(EntType3,EntOcci))

Entity-D-RshipSubset-(RshipType,AnotherRshipType)*

CERTAINTY	certain
FOR	Entity*
ON	deletion
IF	relationship_hidden_entity(RshipType, "Entity*", EntType4, Cmin3,Cmax3), subset_rship((AnotherRshipType,RoleType1,RoleType2), (RshipType,RoleType3,RoleType4)), relationship_participant(AnotherRshipType, RoleType1, Cmin1, Cmax1), foreign_ent_occ(RshipType,Entity*,EntOcc4), which_foreign("Entity*", RoleType3, RoleType4, EntOcci, EntOccj, Entity*, EntOcc4)
ASSERT	¬ (∃ RshipOcc, rship_occ(AnotherRshipType,RshipOcc), rship_occ_part(RshipOcc, RoleType1, EntOcci), rship_occ_part(RshipOcc, RoleType2, EntOccj))
ELSE	reject or propagate(delete(RshipOcc))

Representation 4: Each relationship is represented by a foreign key in different entity types. It covers two possible combinations.

Entity-I-RshipSubset-(AnotherRshipType,RshipType,EntType3)*

CERTAINTY	certain
FOR	Entity*
ON	insertion
IF	relationship_hidden_entity(RshipType, "Entity*", EntType2, Cmin1,Cmax1), subset_rship((RshipType,RoleType1,RoleType2), (AnotherRshipType,RoleType3,RoleType4)), relationship_hidden_entity(AnotherRshipType, EntType3, EntType4, Cmin4,Cmax4), foreign_ent_occ(RshipType,Entity*,EntOcc2)
ASSERT	foreign_ent_occ(AnotherRshipType,EntOcc2,Entity*)
ELSE	reject or propagate(insert(EntType3,EntOcc2))

Entity-D-RshipSubset-(RshipType,AnotherRshipType,EntType1)*

CERTAINTY	certain
FOR	Entity*
ON	deletion
IF	relationship_hidden_entity(RshipType, "Entity*", EntType4, Cmin4,Cmax4), subset_rship((AnotherRshipType,RoleType1,RoleType2), (RshipType,RoleType3,RoleType4)), relationship_hidden_entity(AnotherRshipType, EntType1, EntType2, Cmin2,Cmax2), foreign_ent_occ(RshipType,Entity*,EntOcc4)
ASSERT	¬ foreign_ent_occ(AnotherRshipType,EntOcc4,Entity*)
ELSE	reject or propagate(delete(EntType1,EntOcc4))

Relationships_Union Special SIC:
The actual representations in the relational model are not listed here. We only discuss how many representations there would be. Suppose that we have three relationships $RX = RY \cup RZ$ between entity types E and F. E (or F) cannot have (1,1) cardinalities in both super-relationship (RX) and one of the sub-relationships (RY or RZ). However, it can have (1,1) cardinalities in both RY and RZ. Therefore, there are 17 possible combinations. They can be classified into 7 different sets of SIC representations.

1. Representation 1: All relationships have separate relational representations.
2. The super-relationship is represented by a separate relation, and:
   - Representation 2: Only one sub-relationship is represented by a separate relation. It covers 4 possible combinations.
   - Representation 3: Both sub-relationships are represented by foreign keys in the same entity type. It covers 2 possible combinations.
   - Representation 4: Both sub-relationships are represented by foreign keys in different entity types. It covers 2 possible combinations.
3. The super-relationship is represented by a foreign key, and:
   - Representation 5: Both sub-relationships are represented by separate relations. It covers 2 possible combinations.
   - Representation 6: Only one of the sub-relationship is represented by a foreign key. It covers 4 possible combinations.
   - Representation 7: Both sub-relationships are represented by foreign keys (certainly in the same entity type, but different from the one in which the super-relationship is represented by a foreign key). It covers 2 possible combinations.

Exclusive_Relationship SIC:
There are three sets of representations in the relational model to cover four possible combinations:

Representation 1: Both relationships are represented by separate relations in the relational model. The sub-SICs are the same as those in the E-R-SIC model.

Representation 2: Only one relationship is represented by the foreign key in a non-sharing entity type. It covers two possible combinations.

Relationship-I-RshipExclusive-(SharingEnt1, SharingEnt2, AnotherRshipType, NonShareEnt2)*

CERTAINTY	certain
FOR	Relationship*
ON	insertion
IF	ex_rships(ExRshipSet), belongs_to(("Relationship*",SharingEnt1), ExRshipSet), (¬ relationship_hidden_entity("Relationship*", NonShareEnt1, SharingEnt1, Cmin2,Cmax2)), ∃ AnotherRshipType, AnotherRshipType ≠ "Relationship*", belongs_to((AnotherRshipType,SharingEnt2), ExRshipSet), relationship_hidden_entity(AnotherRshipType, NonShareEnt2, SharingEnt2, Cmin4,Cmax4), ∃ EntOcc2, rship_occ_part(Relationship*, SharingEnt1,EntOcc2)
ASSERT	¬ (∃ EntOcc3, ent_occ(NonShareEnt2,EntOcc3), foreign_ent_occ(AnotherRshipType, EntOcc3,EntOcc2))
ELSE	reject or propagate(delete(NonShareEnt2,EntOcc3))

Entity-I-RshipExclusive-(SharingEnt1,SharingEnt2, RshipType,AnotherRshipType)*

CERTAINTY	certain
FOR	Entity*
ON	insertion
IF	relationship_hidden_entity(RshipType, "Entity*", SharingEnt1, Cmin2,Cmax2), ex_rships(ExRshipSet), belongs_to((RshipType,SharingEnt1), ExRshipSet), ∃ AnotherRshipType, AnotherRshipType ≠ RshipType, belongs_to((AnotherRshipType,SharingEnt2), ExRshipSet), (¬ relationship_hidden_entity(AnotherRshipType, NonShareEntType2, SharingEnt2, Cmin4,Cmax4)), ∃ EntOcc2, ent_occ(SharingEnt1, EntOcc2), foreign_ent_occ(RshipType,Entity*,EntOcc2)
ASSERT	¬ (∃ RshipOcc, rship_occ(AnotherRshipType,RshipOcc), rship_occ_part(RshipOcc, SharingEnt2, EntOcc2))
ELSE	reject or propagate(delete(RshipOcc))

Representation 3: Both relationships are represented by foreign keys in non-sharing entity type(s), respectively.

Entity-I-RshipExclusive-(SharingEnt1, SharingEnt2, RshipType, AnotherRshipType, NonShareEnt1, NonShareEnt2)*

CERTAINTY	certain
FOR	Entity*
ON	insertion
IF	relationship_hidden_entity(RshipType, "Entity*", SharingEnt1, Cmin2,Cmax2), ex_rships(ExRshipSet), belongs_to((RshipType,SharingEnt1), ExRshipSet), ∃ AnotherRshipType, AnotherRshipType ≠ RshipType, belongs_to((AnotherRshipType,SharingEnt2), ExRshipSet), relationship_hidden_entity(AnotherRshipType, NonShareEnt2, SharingEnt2, Cmin2,Cmax2), ∃ EntOcc2, ent_occ(SharingEnt1,EntOcc2), foreign_ent_occ(RshipType,Entity*,EntOcc2)
ASSERT	¬ (∃ EntOcc3, foreign_ent_occ(AnotherRshipType, EntOcc3,EntOcc2), ent_occ(NonShareEnt2,EntOcc3))
ELSE	reject or propagate(delete(NonShareEnt2,EntOcc3))

Totality Constraints:
If in a total constraint the relationship is represented by a separate relation, the sub-SICs are the same as those in the E-R-SIC model. Otherwise, the sub-SICs are represented as follows.

Entity-I-Totality-(RshipType)*

CERTAINTY	certain
FOR	Entity*
ON	insertion
IF	relationship_hidden_entity(RshipType, "Entity*", FEntType, Cmin2,Cmax2)
ASSERT	foreign_key(RshipType, Entity*, FEntType, FKAtts), is_not_null(FKAtts)
ELSE	reject

Entity-I-Totality-(EntType, RshipType)*

CERTAINTY	certain
FOR	Entity*
ON	insertion
IF	relationship_hidden_entity(RshipType, EntType, "Entity*", Cmin2,Cmax2), Cmin2 $\geq$ 1
ASSERT	$\exists$ EntOcc, foreign_ent_occ(RshipType,EntOcc, Entity*), ent_occ(EntType, EntOcc), count(EntOcc) $\geq$ Cmin2
ELSE	reject or propagate(insert_all(EntType,EntOcc))

Entity-D-Totality-(FEntType, RshipType)*

CERTAINTY	certain
FOR	Entity*
ON	deletion
IF	relationship_hidden_entity(RshipType, "Entity*", FEntType, Cmin2,Cmax2), Cmin2 $\geq$ 1, foreign_ent_occ(RshipType, Entity_0^*, FEntOcc), ent_occ(FEntType, FEntOcc)
ASSERT	$\exists$ Entity_1^*, $\text{Entity}_1^* \neq \text{Entity}_0^*$, foreign_ent_occ(RshipType, Entity_1^*,FEntOcc), count(Entity_1^*) $\geq$ Cmin2
ELSE	reject or propagate(delete(FEntType,FEntOcc))

Relative Maximum Cardinality Constraint:

Entity-I-RshipMaxCard-(FEntType,RshipType)*

CERTAINTY	certain
FOR	Entity*
ON	insertion
IF	relationship_hidden_entity(RshipType, "Entity*", FEntType, Cmin2,Cmax2), Cmax2 $\neq$ "*", foreign_ent_occ(RshipType,Entity$^*_0$, FEntOcc), ent_occ(FEntType, FEntOcc)
ASSERT	$\exists$ Entity$^*_1$, foreign_entity_occ(RshipType,Entity$^*_1$, FEntOcc), count(Entity$^*_1$) $\leq$ Cmax2
ELSE	reject