A Logical Analysis of Information Systems: static aspects of the data-oriented perspective

A thesis submitted by

Terence Aidan Halpin
BSc, BA, DipEd, MLitStud

in the

Department of Computer Science
University of Queensland

for the degree of

Doctor of Philosophy

1989 July

Declaration

To the best of my knowledge and belief, the work presented in this thesis is original, except where acknowledged below or in the body of the text. The material has not been submitted, either in whole or in part, for a degree at this or any other university.

This thesis is largely concerned with a version of NIAM (Nijssen's Information and Analysis Method), which I have refined and extended. As the acronym suggests, this methodology was originally developed by Professor G. M. Nijssen. While several researchers have contributed to NIAM, the fundamental conceptual framework was principally the work of Professor Nijssen and Professor E. D. Falkenberg.

The thesis makes substantial reference to work discussed in the first eleven chapters of the text *Conceptual Schema and Relational Database Design* (1989, Prentice Hall, Sydney), which Professor Nijssen and I co-authored. Except for some of the exercise questions, these chapters, and the exercise answers, were written by myself. With respect to original contributions, the following list summarizes the main NIAM enhancements due to myself which are discussed in the above-mentioned text:

- reordering of the steps in the Conceptual Schema Design Procedure to improve the treatment of subtypes and mandatory roles;
- extensions to uniqueness constraints for nested fact types;
- explicit distinctions between populations and types, and interactions between real world and data base constraints;
- simplification of subtype treatment and removal of previous anomalies;
- extensions to occurrence frequencies and label type constraints;
- deeper analysis of reference schemes;
- simpler notations for equality, subset and exclusion constraints;
- various constraint implication theorems;
- additional constraint types (irreflexive, asymmetric, intransitive, mandatory entity);
- deeper analysis of derivation, and discussion of open/closed worlds;
- additional results concerning conceptual schema equivalences;
- schema transformation algorithm based on degree of overlap;
- notations for constraints on relational schemas;
- adding comprehensive constraint mapping to the ONF algorithm;
- conceptual pre-optimization to obtain a better ONF relational schema.

TA Halpin

Terry Halpin

Brisbane, 1989 July

Acknowledgements

I am grateful to my supervisor, Professor John Staples, for his sage advice, encouragement, kindness, and meticulous appraisal of earlier drafts of this thesis.

Professors G.M. Nijssen and E.D. Falkenberg have my thanks for many stimulating discussions on NIAM. Apart from providing conceptual foundations, their communication skills have helped me to appreciate the value of examples and diagrams.

Peter Creasy has my gratitude for reading and commenting on the second draft of the thesis. I also thank Lee Lafferty for writing the printer driver which facilitated the production of the printed copy.

Finally, I thank my wife, Norma, for her love, support and patience.

Abstract

It is widely accepted that information systems are best specified first at the conceptual level. This approach promotes correctness, clarity, adaptability and productivity. For commercial applications, relational database systems have become the most important target systems for implementing conceptual information structures. This is mainly because relational systems are simpler to use, are based on mathematical foundations, and are now efficiently implemented. Designing appropriate conceptual and relational schemas for practical applications is a non-trivial task. The main objective of this thesis is to provide a formal basis for reasoning about conceptual schemas and for making design choices.

This thesis focusses on the data-oriented perspective of information system design. Most conceptual modelling methodologies provide a graphical language for the high level specification of conceptual schemas. Of these graphical notations, the conceptual schema diagram language of NIAM (Nijssen's Information Analysis Method) is arguably the most intuitive and expressive. Partly because of these advantages, the designer may experience difficulties in assessing various properties of and relationships between NIAM conceptual schemas (e.g. satisfiability, implication and equivalence). To help resolve these difficulties, we formalize NIAM conceptual schemas in terms of formal logic, so as to provide a rigorous treatment of the formal semantics and proof-theory.

A thorough analysis of database reference schemes is provided, including definite descriptions. Global and local aspects of schemas are distinguished, so as to support modular specification. Various results concerning derivation rules are established, and NIAM is extended to take full advantage of the semantics of numeric and lexical objects. Further extensions and modifications to NIAM are introduced, motivated and formalized.

Properties of, and relationships between conceptual schemas are treated in depth, with particular attention to satisfiability, constraint implication, and schema equivalence. One of the major contributions of this thesis is its rigorous formalization of schema equivalence within NIAM, enabling transformation theorems to be precisely stated and formally proved. The formalism is used to refine existing results and to establish new theorems.

The implementation of conceptual schemas in relational database systems is examined, focussing on the mapping of conceptual constraints into relational schemas and SQL systems in particular. One important application discussed is the notion of conceptual optimization, whereby conceptual schemas are transformed before being mapped down, in order to yield a more efficient relational schema.

Related topics for future research are suggested. Appendices provide background on formalization, examples of detailed formal proofs, and indicate how some of the results developed here may be adapted to the Entity-Relationship modelling approach.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Thesis scope and motivation

In this section we briefly outline the scope and motivation of the thesis. The next section summarizes the main contributions of this work, and the final section of the chapter provides a structural overview.

This thesis is largely concerned with the formal specification of conceptual structures, and their mapping to relational database systems. Although much of what is presented here has relevance to the design of conceptual and relational schemas in general, the focus of the conceptual modelling treatment is on NIAM (Nijssen's Information Analysis Method), with the relational code being given in SQL (Structured Query Language).

For reasons such as correctness, clarity, adaptability and cost-effectiveness, information systems are best specified first at the conceptual level (ISO 1982). For commercial applications, relational database systems have generally become the most important target systems onto which conceptual information structures are mapped. This is mainly because relational systems are simpler to use, are based on mathematical foundations (Codd 1970), and are now efficiently implemented. Moreover, as Codd points out, "the relational model ... extends itself naturally to distributed systems ability to extract the information" (Rapaport 1988).

Designing appropriate conceptual and relational schemas for practical applications is a non-trivial task. Each year thousands of research papers appear in the literature dealing with aspects of this design problem. The main objective of this thesis is to provide a formal basis for reasoning about conceptual schemas and for making design choices which lead to efficient relational implementations.

Many conceptual modelling methodologies exist (e.g. see Brachman 1988; Jardine & Reuber 1984; Olle et al. 1988; Sowa 1988). Although the process-oriented and behaviour-oriented perspectives of information system design are important, this thesis focusses on the data-oriented perspective. One popular way to design relational data structures is to use entity relationship modelling (ER) to specify the conceptual schema, map this onto a relational schema, and then refine the table structure using normalization (e.g. Chen & Dogac 1983; Teorey, Yang & Fry 1986). Although similar to ER modelling in

some respects, fact-oriented modelling, as exemplified by NIAM, arguably provides a simpler and stronger methodology (see Appendix 3 for further discussion of ER).

Comparative benefits of the fact-based approach have been cited elsewhere (e.g. Kent 1986; Nijssen, Duke & Twine 1988). Natural verbalization of examples in terms of elementary facts is the foundation of NIAM's design procedure. Its conceptual schema diagrams use only one data structure (the fact type), allow a wide variety of constraints to be expressed, and are easily populated for validation purposes.

Originally developed by Nijssen, Falkenberg and others, NIAM has evolved considerably over the last few years. Papers by Falkenberg (1976) and Vermeir & Nijssen (1982) illustrate an early form of NIAM which adopted the Binary Relationship Model. To provide a more natural and direct connection with human conceptualization, NIAM now allows relationships of any arity. For historical background on the binary-relationship model, as well as an argument for retaining the binary-only restriction, see Mark (1987). A recent text (Nijssen & Halpin 1989) provides a detailed introduction to the version of NIAM currently endorsed by Nijssen.

Given a (hopefully) significant set of output reports for a UoD (Universe of Discourse), NIAM's design procedure is used to specify a conceptual schema in which each fact type is (ideally) elementary. For implementation in a relational database system, a simple algorithm is then used to group these fact types into relation types in "optimal normal form" (ONF). The number and shape of tables so obtained can often be changed by transforming the initial conceptual schema into another which is equivalent (or acceptably close to being equivalent: we refine this notion later) before applying the ONF algorithm. Hence such transformations can help to produce a more efficient relational schema which is still free of redundancy. Conceptual schema transformations also play an important role in the merging of subschemas and translation between different user views.

Most conceptual modelling methodologies provide a graphical language for the high level specification of conceptual schemas. Of these graphical notations, the conceptual schema diagram language of NIAM is perhaps the most intuitive and expressive. However, as pointed out by Levesque (1984), there is a fundamental tradeoff between expressibility and tractability. The more expressive a language becomes, the greater the computational complexity of procedures for checking logical results (e.g. constraint implication, or schema equivalence).

Partly to keep the problem manageable, most research in database design has restricted the set of constraints to functional and multi-valued dependencies (e.g. see Beeri & Kifer 1986). Nevertheless, in practical database applications the additional constraint categories depicted on NIAM diagrams often occur, and hence should not be ignored. Partly because NIAM schemas are so expressive, the designer may experience difficulties in assessing various properties of and relationships between NIAM conceptual schemas, e.g. satisfiability, implication and equivalence.

To help resolve these difficulties, we formalize NIAM conceptual schemas in terms of first order predicate logic, thereby providing a rigorous and well-founded treatment of the formal semantics and proof-theory. Although formalization of knowledge bases in terms of formal logic is not new (e.g. ISO 1982 appendix F, Reiter 1984), we are not aware of any previous formalization of NIAM in predicate logic. Our formalization in first order logic supports the rigorous proof of substantial practical results (e.g. schema equivalence). It also provides a simple and natural treatment of various theoretical aspects (e.g. definite descriptions in the database context).

This logical framework is used to refine and extend NIAM in several ways. New results are obtained in the following areas: reference schemes; derivation rules; lexical and numeric aspects; further constraint categories; schema satisfiability; constraint implication; schema equivalence and schema implication; conceptual optimization; and constraint mapping to ONF relational schemas.

Many of the problems addressed in this thesis overlap with the problems identified by Gallaire, Minker and Nicolas (1984, p. 179) as needing continued research. We hope that our work further demonstrates the fruitful interaction of the disciplines of logic and database theory.

1.2 Summary of main contributions

The major research contributions of this thesis may be summarized as follows. They are listed roughly in decreasing order of importance, from our perspective. This does not correspond to the order in which the contributions are discussed in the thesis.

1 Formalization of NIAM in terms of first order predicate logic, thus providing a well-founded model theory and proof theory.

2 A rigorous account of the notions of equivalence and implication between conceptual schemas, as well as constraint implication, thus allowing relevant theorems to be precisely specified and proved.

3 New theorems concerning constraint implication, and conceptual schema equivalence and implication in NIAM.

4 A proper formal account of NIAM reference schemes, with particular attention to definite descriptions and lexical/numeric objects.

5 Guidelines for optimizing the conceptual schema by transforming it to yield a more efficient ONF map.

6 Augmentation of the ONF algorithm by adding comprehensive mapping of conceptual constraints to relational constraints.

7 Separation of local/global aspects of conceptual schemas to support incremental specification and general application of results.

8 Deeper analysis of derivation rules, including specification and use of semantics for lexical and numeric object types.

9 A rigorous account of conceptual schema satisfiability in NIAM.

10 Various early refinements and extensions to NIAM, including:

- reordering of the steps in the Conceptual Schema Design Procedure to improve the treatment of subtypes and mandatory roles;
- detection and removal of anomalies in the treatment of subtypes;
- explicit distinctions between populations and types, and interactions between real world and data base constraints;
- additional constraint categories;
- extensions to uniqueness constraints for nested fact types;
- extensions to frequency and lexical/numeric constraints;
- simpler notations for equality, subset and exclusion constraints.

1.3 Structural overview

We conclude this introductory chapter by providing a structural overview of the thesis content. A perusal of the Contents pages will help to fill in further details of the development. Please note that within this thesis, unless otherwise specified, plural first person terms (e.g. "we", "our") are to be interpreted as singular (e.g. "I", "my").

As indicated in the thesis declaration, we recently co-authored a NIAM textbook (Nijssen & Halpin 1989). For convenience, from now on we refer to this work as *NH89*. In chapter 2 of this thesis, we summarize some of our main enhancements to NIAM as presented in NH89, focussing on the examples cited in the previous section under contribution 11. The early placement of this material reflects the chronology of its development, and provides an opportunity for informal discussion of concepts to be later formalized.

In Chapter 3, we define our usage of some general terms (e.g. "universe of discourse", "knowledge base"), and discuss a simple conceptual architecture for information systems. We then present the syntax and formal semantics of the knowledge base language (KL) that will be used in our formalization. Generic axioms are specified to partition the domain of discourse and to provide semantics for real numbers and character strings. Some basic theorems are established from these axioms.

The next two chapters specify how knowledge bases expressed in NIAM's high level graphic and tabular notation may be translated into sentences in KL. Additionally, several revisions and extensions are made to improve various aspects of NIAM. Chapter 4 covers the basic issues involved in specifying object types, predicates and constraints.

Chapter 5 begins with a detailed analysis of reference schemes, especially those involving numbers. It then examines those aspects of conceptual schemas which require a global perspective. An examination of derivation rules follows; as a result these rules are partitioned into two categories. Finally, a theory of definite descriptions is presented that enables specific databases to be mapped into KL.

Chapter 6 uses the formal framework to analyze various modal properties of and relationships between conceptual schemas. A stronger notion of satisfiability is introduced for schemas, and additional schema formation rules are specified. Constraint implication is then defined and several theorems in this category are established. The final section of this chapter deals with equivalence and implication between conceptual schemas. This is

probably the most important part of the whole thesis: by introducing contextual definitions which provide conservative extensions, the notion of schema equivalence is rigorously grounded in first order logic, and several important theorems are proved.

Chapter 7 examines various applications of the theory to the implementation of conceptual schemas in relational database systems. To complement the ONF algorithm, a procedure is discussed for mapping conceptual constraints into relational schemas and SQL systems in particular. The notion of conceptual optimization is examined: to generate a more efficient relational schema, a conceptual schema may be transformed before being input to the ONF algorithm. Guidelines for selecting conceptual schema transformations are discussed.

Chapter 8 summarizes the principal ideas in the thesis, and suggests related topics for future research.

There are three appendices. Appendix I provides some basic background on formalization in general. Appendix II provides sample formal proofs of some of the results discussed in the thesis. Appendix III briefly indicates how some of our results may be adapted for use with Entity-Relationship modelling.

2 Early enhancements to NIAM

2.1 Design sequence and subtyping

In this chapter we briefly summarize some of our early enhancements to NIAM, as incorporated in NH89. Many of these changes were also discussed in our earlier works (e.g. Halpin 1986a, 1987, 1988a, 1988b). As well as having practical advantages, these enhancements are a preparation for our later formalization. In this section we focus on modifications we made to the NIAM *Conceptual Schema Design Procedure* (CSDP) to overcome some problems with the treatment of subtypes and mandatory roles. A brief sketch of NIAM is included, but the reader unfamiliar with this methodology may wish to consult NH89 for further background.

Basically, NIAM is a method of designing information systems. Though it includes some mechanisms for specifying information flows (information flow diagrams) and modelling an application's behaviour (event triggers), its emphasis is on the design of information structures. In particular, NIAM provides a procedure (the CSDP) for specifying information structures at the conceptual level, as well as an algorithm (the ONF or "Optimal Normal Form" algorithm) for mapping these conceptual structures onto normalized relational schemas for implementation in relational database systems. The conceptual schema design procedure currently comprises nine steps (see Table 2.1).

1. Transform familiar information examples into elementary facts, and apply quality checks.
2. Draw a first draft of the conceptual schema diagram, and apply a population check.
3. Eliminate surplus entity types and common roles, and identify any derived fact types.
4. Add uniqueness constraints for each fact type.
5. Check that fact types are of the right arity.
6. Add entity type, mandatory role, subtype and occurrence frequency constraints.
7. Check that each entity can be identified.
8. Add equality, exclusion, subset and other constraints.
9. Check that the conceptual schema is consistent with the original examples, has no redundancy, and is complete.

Table 2.1 The conceptual schema design procedure (CSDP)

Prior to our work, the CSDP comprised 14 steps, in which subtypes were determined at step 5, uniqueness constraints at step 6, and mandatory (total) roles were specified at step 8 (Nijssen & Falkenberg 1983). In NIAM, unlike many approaches to subtyping, a subtype is introduced only if the following two conditions are satisfied: some role is played only by this subtype; the subtype is definable in terms of other roles played by its supertype(s). So a subtype is introduced for a particular role only if this role is optional for its attached object type. Hence, contrary to the above ordering of steps, we need to determine whether the role is optional or mandatory before we can make any decision about subtyping.

Since the old order of the CSDP steps is inappropriate, we changed the CSDP to have mandatory roles determined before subtyping. Partly to simplify the specification of exclusion and exhaustion constraints among subtypes (see later), we also demanded that uniqueness constraints and label type constraints be determined prior to subtyping. By making some other minor changes and combinations, we reduced the CSDP to 9 steps (NH89, p. 32). The subtype introduction procedure was reworded to ensure that only well-defined proper subtypes could be introduced (NH89, p. 132).

Although we believe the new order of the CSDP steps is appropriate for *learning* the design methodology, in actually *using* the procedure an experienced designer may choose to reorder some of these steps or to perform some steps concurrently. For example, the candidate elementary facts verbalized at step 1 might in fact be splittable; rather than waiting till step 5 to apply formal checks on these candidates, the designer may well decide to consider uniqueness constraints (steps 4 and 5) concurrently at step 1 in order to minimize the chance of making such errors before actually drawing a draft diagram. Knowledge of uniqueness constraints may also impact on decisions about derived fact types (step 3). For example, if we agree that constraints on derived fact types should also be derivable then cases traditionally considered in terms of transitively implied functional dependencies are readily identified. Later constraints (especially mandatory roles, subset and equality constraints) may also impact on decisions about derivability.

Owing to the work of Falkenberg, Nijssen and Vermeir, NIAM includes two matrix procedures for determining the subtype graph (Vermeir & Nijssen 1982, Nijssen & Falkenberg 1983). There has been a tendency in the NIAM literature to suggest that all aspects of a conceptual schema, including subtyping, can be determined from a significant set of output reports. However, as we

indicated in NH89 (p. 135), no set of reports can be significant with respect to subtype definitions, since for any finite set of data there will always remain an infinite set of such definitions which are consistent with the data. The same comment applies to derivation rules. In cases of doubt, the UoD expert must be consulted.

NIAM uses an arrow between nodes to indicate that the source node is a proper subtype of the target node. During the course of its evolution, NIAM has used various additional notations to specify exclusion and exhaustion constraints among subtypes. The notation used prior to our work is shown in Figure 2.1. This scheme was developed by Nijssen and Falkenberg (1983). In cases 1 and 3 the dot indicates that, for each state of the knowledge base, the populations of the subtypes must collectively exhaust the population of the supertype. In cases 1 and 2 the fork indicates that the subtypes are disjoint; in cases 3 and 4 the subtype populations may overlap.

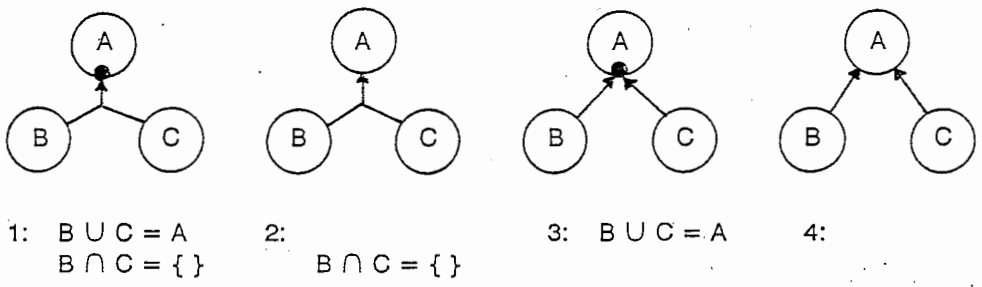
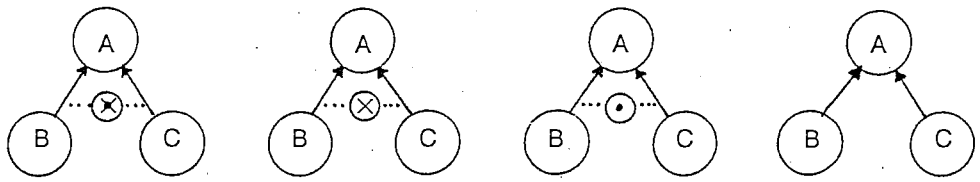


Figure 2.1 An earlier graphical notation for subtypes

To provide some intuitive support for this notation, it was argued that separate arrows suggests different classification schemes, which result in overlapping subtypes. However, different classification schemes need not overlap. For example, consider a chauvinistic universe of discourse in which all high salary earners are males. Now suppose number of children is recorded only for female employees, and home phone number is recorded only for high salaried employees. We now have two disjoint subtypes based on different classification schemes (gender and salary).

A more serious problem with the scheme is its inability to fully specify exclusion constraints for various cases when more than two subtypes are involved. For example, suppose A has subtypes B, C and D, and types C and D overlap with each other but not with B. For instance, suppose we wish to record some specific role for each of the subtypes Non-smoker, Pipe-smoker and Cigarette-smoker, where the latter two types overlap. The scheme in Figure 2.1 does not extend to handle such cases.

For such reasons we abandoned the notation. In response to our criticisms, Falkenberg (1986) now uses a notation in which exclusion and exhaustion between two subtypes is specified by a circled dot and circled "×", respectively, connected to the subtype arrows by dotted lines (see Figure 2.2). This notation is more intuitive, and is capable of specifying the cases where the old notation failed.



1: B ∪ C = A
B ∩ C = { }

2:
B ∩ C = { }

3: B ∪ C = A

4:

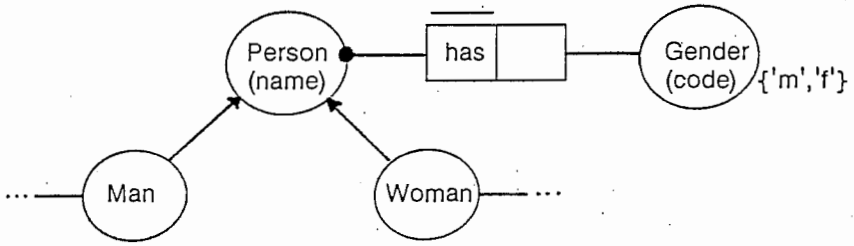
Figure 2.2 The notation now used by Falkenberg

However, this notation leads to untidy diagrams. When the subtype graph has several nodes, there may be so many exclusion and exhaustion markers crowded into the schema diagram that the diagram is difficult to read. Since a major reason for using a diagram is to provide a clear picture, we feel that extensive use of this notation is less than ideal.

There are in fact two basic purposes served by a conceptual schema diagram: to provide a concise means of quickly expressing most of the conceptual schema; and to provide a simple picture of the application for humans. With this in mind, a solution to the problem suggests itself. Just as derived fact types must be fully specified by textual formulae, so must subtype definitions. And just as constraints on derived fact types may be omitted from the diagram since they are implied, subtype exclusion and exhaustion constraint markers may be omitted since they are implied by the subtype definitions and the constraints on the fact types involved in these definitions.

So we changed the CSDP to require that proper definitions be supplied for subtypes when they are introduced, and to remove the requirement for marking subtype exclusion and exhaustion constraints. Another reason for reordering the steps was to expose the connection between these implied constraints, the subtype definitions and the current constraint picture. In rare cases, as discussed shortly, other constraint types may be required to complete this picture. In the absence of explicit constraint markers, it is important for the designer to choose subtype names which suggest these constraints in the

mind of the human reader. As a simple example, consider the subschema shown in Figure 2.3. For simplicity, the roles played by the subtypes are omitted, and a high level language for subtype definition has been assumed.



Man $=_{df}$ Person having Gender with code 'm'
Woman $=_{df}$ Person having Gender with code 'f'

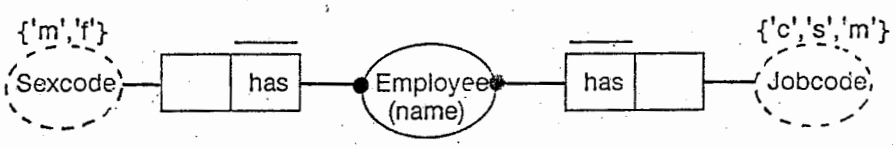
Figure 2.3 It is implied that Man and Woman partition Person

The dot and arrowed bar respectively specify mandatory role and uniqueness constraints (in combination they assert that each person has exactly one gender). The constraint that Man and Woman are mutually exclusive is implied by the subtype definitions, together with the uniqueness constraint (each person has at most one gender) and the fact that 'm' is not equal to 'f'. The constraint that the union of Man and Woman exhausts Person follows from the subtype definitions, the lexical constraint ({'m','f'}), and the mandatory role constraint on has_gender. So there is no need to clutter up the diagram by marking these constraints.

Later in the thesis we formalize the notion of constraint implication and provide a more rigorous discussion of such cases. The relevant theorems for this case are IX2 and IE1 (see section 6.2). In setting out these theorems we make use of Falkenberg's notation, to pictorially convey the implied constraints. For such abstract situations the notation is still of value. In very rare cases, the notation may also be useful in a specific situation, as we now discuss.

Subtype exclusion and exhaustion constraints are always implied by the subtype definitions and the constraints on the defining fact types. In practical cases, the constraints of relevance here tend to be just uniqueness, mandatory role and lexical/numeric constraints. One can imagine rare cases however, when exclusion or exhaustion constraints may need to be specified directly. Since such a case may arise even if subtypes are not introduced, our general solution to this problem is to specify these as *textual constraints*, written beneath the schema diagram.

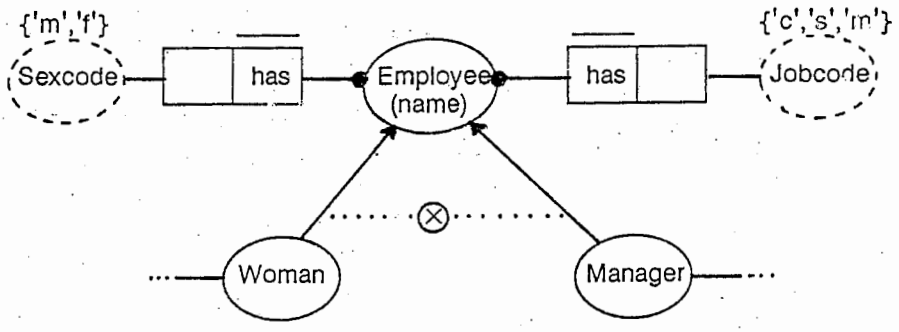
Since such cases are hard to invent, we provide only one, rather pathological example. Figure 2.4 describes a chauvinistic company whose employees hold the position of clerk ('c'), secretary ('s') or manager ('m'). Here it is company policy that no women are managers. For simplicity we have specified two of the employee attributes directly in terms of codes, rather than introduce Sex and Job as entity types with codes. The details of the textual constraint (symbolized below the braced comment), may be ignored. A formal treatment is given in later chapters.



{ No women are managers }
∀x~(x has_sexcode 'f' & x has_jobcode 'm')

Figure 2.4 An "exclusion constraint" specified textually

However, if subtypes are introduced (in NIAM this happens only if the subtypes have specific roles to play), then Falkenberg's notation may now be used as a simple, graphic way of portraying these constraints (see Figure 2.5). For an exhaustion case, add the constraint that all men are managers. We allow the graphic notation in such rare cases. In other cases, except for discussion of constraint implication, the graphic notation for subtype exclusion and exhaustion is best ignored.



Woman $=_{df}$ Employee having Sexcode 'f'
Manager $=_{df}$ Employee having Jobcode 'm'

Figure 2.5 An exclusion constraint specified graphically

2.2 Constraints

In this section we summarize a number of changes and additions that we made to the treatment of constraints in NIAM conceptual schemas. These are discussed more fully in NH89.

Although the conceptual schema is typically used to specify some real world application, we may have good reasons for allowing the set of constraints imposed on the *knowledge base* (the formal model of the application) to differ from the actual constraints in the real world. However, some general results may be stated which restrict the possible relationships between *real world constraints* and *knowledge base constraints*. We discuss two such results here, by way of example.

Consider the subschema of Figure 2.6. For simplicity, reference modes have been omitted. The constraints indicate that each person has exactly one year of birth, and zero or more phones. The same phone may belong to more than one person. Let us suppose that this constraint picture agrees with the real world.

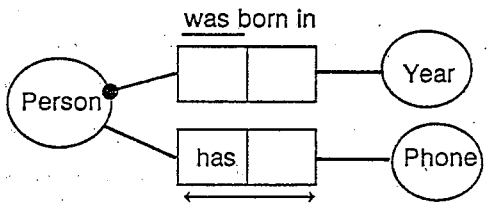


Figure 2.6 A subschema for the real world

Now for good reasons (e.g. privacy, space limitations) we may decide to make the recording of birth year optional, and to record at most one phone number for each person. So the corresponding subschema for our knowledge base is as shown in Figure 2.7.

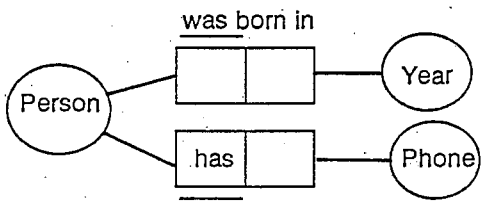


Figure 2.7 A different subschema for the knowledge base

Since we usually make the knowledge base constraints agree with those in the real world, if there are any differences in the constraint patterns these

should be consciously determined by the schema designer. The following guidelines need to be observed in specifying constraints on the knowledge base.

1 For each fact type, its pattern of uniqueness constraints needs to be at least as strong as that which applies in the real world.

2 If a role is optional in the real world, then it is optional in the knowledge base; but the converse need not apply.

Note that these guidelines are observed in our example. The first guideline would be violated if we weakened the uniqueness constraint on was_born_in. We do not wish to allow a person to have many birth years (at least not in the same incarnation!). The second guideline would be violated if we made it mandatory for a person to have a phone. We can't record a person's phone if the person doesn't have one (of course we can always change the situation by demanding that people must have phones in the real world; but this is a different issue). For further discussion, see NH89 (pp. 77, 117).

In NH89 we introduced some new constraint categories. Three of these (*irreflexivity*, *asymmetry* and *intransitivity*) relate to fact types where at least two of the roles are played by the same object type. The most common application is the homogeneous binary fact type. For example, the parenthood relation is irreflexive (nobody is his/her own parent), asymmetric (if x is a parent of y then y is not a parent of x), and intransitive (if x is a parent of y and y is a parent of z then x cannot be a parent of z). These constraints are indicated by marking "ir", "as" or "it" near the predicate (see Figure 2.8). Since irreflexivity is implied by asymmetry, the former constraint is omitted.

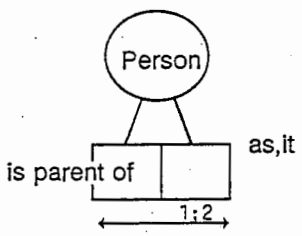


Figure 2.8 The parenthood relation

While such constraints are standardly included in the set of relational properties discussed in introductory logic, they are often ignored in the database context. This is a pity, since they can be implemented quite

efficiently (as discussed in a later chapter). For further discussion see NH89 (pp. 183-191), and for a formal treatment see chapter 4 of this thesis.

One other constraint we introduced in NH89 (p. 195) is the "*mandatory entity*" constraint, to cover cases where a specific object has to be present in any population of its object type (e.g. a file-server node might be mandatory for any network of nodes). We have now subsumed this category under a wider class of constraints which we call "role-object constraints". These are discussed later (chapter 4).

We now mention one subtle change we made to the notion of *occurrence frequency constraints* in NIAM. These are discussed in NH89 (pp. 149-154) and formally defined in chapter 4, and may apply to a single role or a combination of roles. In Figure 2.8, the mark "1;2" denotes a frequency constraint on the second role of the is_parent_of predicate (we no longer use the "1..2" notation of NH89). This means that each person that appears in the population of this role must appear there either once or twice (each person has at most two parents).

Nijssen and Falkenberg (1983) instead defined an occurrence frequency to be the number of times an object in the attached object type could play that role. Their definition has two unfortunate consequences. Firstly, it introduces a dependence between mandatory roles and frequency constraints. For example, if a role is optional the lower bound on its frequency must be zero, but if the role is mandatory this lower bound must be above zero. With our approach the two kinds of constraint can be treated independently.

Secondly, their notation is less powerful, since it cannot be used to specify frequency constraints on optional roles when the lower frequency bound exceeds zero. For example, using our notation the "2" mark in Figure 2.9 specifies that if a person has a phone at all, he or she must have 2 phones. This constraint cannot be expressed in the old notation.

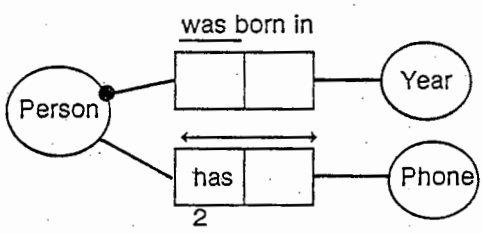


Figure 2.9 A simple frequency constraint on an optional role

For such reasons we changed the definition of the term "occurrence frequency constraint" to the definition given here. This also made it easier

to specify general constraint implication theorems dealing with frequency constraints (see NH89, pp. 152-3, as well as section 6.2 of this thesis). We have since found another version of NIAM in which a definition of occurrence frequency equivalent to ours is used (Mark, 1987).

In NH89 we introduced a number of extensions to standard NIAM constraints. For example, we allow that some of the roles spanned by an inter-predicate uniqueness constraint may belong to an objectified relation type. For example, the circled "u" in Figure 2.10 captures the constraint that for each subject and position there is only one person enrolled in that subject who achieves that position. The *rounded rectangle* around the enrolled_in predicate is our new notation for indicating nesting: it implicitly specifies that this predicate is many:many. The objects playing the achieves role are (Person,Subject) pairs nested in the enrolment fact type. We discuss nesting in detail later. For further discussion on this example using the old notation see NH89 (pp. 88-9).

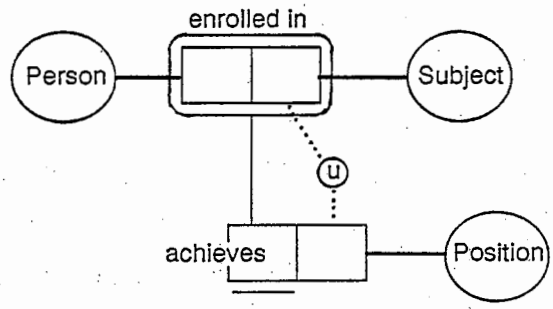


Figure 2.10 A needed uniqueness constraint

Unless such constraints are allowed, there is no way of formally supporting any notion of equivalence for the nesting/flattening transformations, since some uniqueness constraints in the flattened version could not be expressed in the nested version. We discuss the general formalization of schema equivalence in detail later in the thesis.

We also extended the varieties of "label type constraints" to include lexical and numeric subtypes of several forms (e.g. see NH89 pp. 111-3). However, since we later (section 4.3) provide an improved and more comprehensive treatment of this area, we say no more about this now.

We conclude this section by noting some simplifications we made to the notations for *subset, equality and exclusion constraints*. The new notations are discussed in NH89 (pp. 171-81) and are formally defined later in the thesis (chapter 4). Basically, we use a dotted arrow for a subset constraint, running from the subset role to the superset role. For an equality constraint

there are arrow heads at both ends. Apart from being simpler, this notation suggests the close connection between a subset constraint and a conditional.

Exclusion constraints are specified by an "×" as usual, but without the circle (unless there are more than two operands). When the roles are contiguous, the dotted lines should meet at the junction point (this avoids the need for role connectors at each end). Figure 2.11 gives one example of the old versus the new notation.

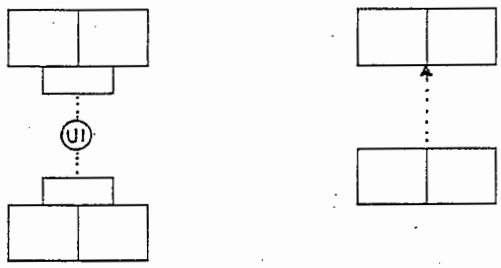


Figure 2.11 Old and new notations for a pairwise subset constraint

2.3 Other enhancements

In NH89 we introduced several other enhancements to NIAM. These include a deeper analysis of reference schemes, an overlap algorithm for selecting schema transformations based on the kind of relation overlap, new results for constraint implication and conceptual schema equivalence, high level notations for constraints on relational schemas, augmentation of the optimal normal form algorithm by constraint mapping, and conceptual schema optimization to provide a more efficient ONF map.

Some of these ideas have now been substantially reworked and improved (e.g. our treatment of reference schemes and schema equivalence). Moreover, these contributions are best appreciated once a formal groundwork for them has been set out. Hence, rather than examining these matters at this stage, we postpone their discussion until our basic formalization of NIAM has been presented. The next three chapters provide this formalization.

3 Formalization of information structures

3.1 The UoD and the knowledge base

In this chapter we provide a framework for formalizing information structures; the concepts and definitions presented are used later to establish various results, especially theorems about constraint implication and schema equivalence. In this section we provide an intuitive explanation of some fundamental aspects of our approach to *information systems*. A more formal treatment is provided in later sections. We use the term "information systems" in a generic sense, to include not just traditional database systems but more advanced systems such as "knowledge-based systems".

Though they can be treated formally, information systems are constructed for pragmatic reasons. To begin with, there is a *task* that is required to be carried out with the aid of a computer system, e.g. handling of academic records. The complete specification of the task/problem/application may involve data, process and behavioural aspects (see Olle et al. 1988). We ignore the latter two perspectives for this thesis, and assume that the problem is solved if the system can output on request any *information* perceived to be relevant.

Strictly, and conceptually, the information system (IS) stores *sentences* which are interpreted by the user as expressing information. We assume the reader is familiar with the notion of external, conceptual and internal levels (e.g. see ISO 1982). Our discussion remains at the conceptual level unless otherwise indicated.

Such information involves a fixed area of interest known as the *universe of discourse* (UoD). In the research literature, the term "universe of discourse" has many meanings (see, e.g., ISO 1982; Jardine & Reuber 1984). Our usage is now explained. The UoD is typically specified with the hope of modelling a very restricted structure within the physical universe (e.g. a business environment), but may specify an abstract world (e.g. Euclidean geometry) or an imaginary world (e.g. Wonderland).

To clarify our intuitive usage of the term "UoD" we make use of the notion of *possible worlds*. The nature of possible worlds and the status of transworld identity of individuals are still areas of philosophical dispute. Possible worlds are construed variously as maximally consistent sets of

propositions, maximally possible states of affairs, alternative conceptions, and so on. Loux (1979) and Haack (1978, pp. 170-203) present useful overviews of the central issues. Hughes and Cresswell (1968) provide a standard introduction to modal logic, while Bradley and Swartz (1979) give an extensive account of modal propositional logic. We make no attempt here to solve the philosophical problems of possibilia semantics.

To avoid circularity, the term "possible world" is understood through paradigm examples. One possible world is the real world. The term "*real world*" or "*actual world*" is usually taken to refer to our whole universe, including past, present and future. One might think of the real world as a space-time continuum. Besides the real world there are infinitely many worlds which might have been, e.g. a world in which Expo88 was held in Sydney rather than Brisbane.

Some worlds are *impossible*, e.g. a world in which Expo88 was held in Brisbane but was not held in Brisbane. We use "possible" in a logical rather than a physical sense, e.g. it is logically possible for us to float in the air even though our physical laws might prevent this.

A *subworld* may be "part" of a world, e.g. the space-time worm which is Australia in the twentieth century. More generally, a subworld may be pictured as a world viewed through a "relevance filter" which removes unwanted detail. Over the lifetime of the application, a particular subworld is modelled in the information system. At any point in this lifetime, the information system provides a "*glimpse*" of the subworld. Each subworld glimpse portrays a set of individual objects instantiating certain properties and relations.

For a given task we associate exactly one UoD. UoDs for different tasks may overlap. We think of the UoD essentially as the *structure* of interest. Hence, extensionally, we may regard each UoD as a *set* of possible subworlds. For a given UoD, each subworld must admit only certain *types of individual*. For example, suppose the task is simply to maintain details concerning the years in which scientists were born or died. In this UoD each individual is either a scientist or a year: there are no cats, computers, languages etc.

Secondly, each subworld must admit only certain *predicates*. We use the term "predicate" to mean "property, or relationship type". In our example UoD, the predicates denoted by the predicate symbols "was_born_in" and "died_in" are the only ones admitted. Thirdly, each subworld must satisfy certain *constraints* on its predicate populations (e.g. each person was born in only one year). Some predicates may be derived from others by means of

derivation rules. For example, given any individual x, x is_dead if and only if there is a y such that x died_in y.

A subworld is *factual* if and only if each relevant atomic proposition true in it is also true in the actual universe; otherwise the subworld is *fictional*. We take a *proposition* to be what is asserted when a declarative sentence is uttered. Propositions are true or false but not both; however in deciding whether a subworld is factual we exclude irrelevant propositions, i.e. those not "of interest" for the UoD. An *atomic* proposition predicates a property or relationship on a sequence of individuals. For a given UoD, all its member subworlds are possible but some of these may be fictional. Figure 3.1 illustrates this classification, with rectangles depicting some sample subworlds.

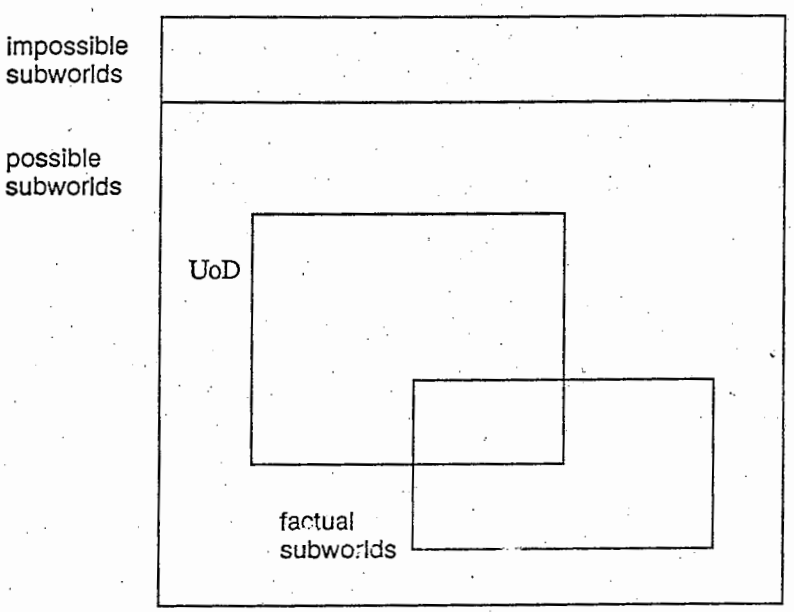


Figure 3.1 A given UoD is a set of possible subworlds

Suppose the task is to maintain birth and death details about scientists. In some impossible subworld, Einstein died in 1955 and did not die in 1955. In some possible, fictional nonUoD subworld Einstein invented Pascal. In some factual (and hence possible) nonUoD subworld, Wirth invented Pascal. In some fictional subworld within the UoD, Einstein died in 1960. In some factual UoD subworld, Einstein died in 1955.

Note that truth in a subworld does not imply *actual truth* (i.e. truth in the real world). In some fictional subworld, the actually false proposition that Einstein died in 1960 is assigned the value true.

In possible world semantics, it is usually agreed that in each possible world *every* proposition is true or false. In standard closed world semantics a proposition is false if it is not asserted to be true. In our subworld semantics every proposition *of interest* is true or false in each subworld. If a proposition is not of interest, it does not concern us what its truth value might be: we simply reject it from consideration. Pragmatically therefore we have no need to decide whether propositions outside the UoD (e.g. Einstein was a physicist) should be assigned "false", "unknown" etc.

Constraints imposed on the UoD may allow some real world instances of a relevant predicate to be omitted (e.g. we might decide to record only one phone number for each person, even though in the real world some people have more, and we might make recording of a person's birth year optional even though each person has a birth year).

For a given UoD there may be several factual UoD subworlds. This might arise because the UoD subworlds relate to different real subworlds, e.g. the same academic record system might be used in two different universities. Note that we consider the UoD as a structure, rather than a structure instance (i.e. a subworld). Two universities may have the same academic recording structure, even though many of their individuals (e.g. students) differ.

Even if the real subworld is the same, UoD subworlds may differ according to how *complete* their details are. For example, suppose the UoD allows that people may play zero, one or more sports, and that in the real world Ann plays tennis and hockey. In one factual UoD subworld we might record no sport for Ann, in another only tennis, in another only hockey, and in another both tennis and hockey.

For a given application there will be one UoD subworld that is *factual and complete*: this is the *task subworld*. Given the UoD specification, the information system can ensure that the particular subworld being described is consistent with this specification. However it is beyond the scope of the system to ensure that this subworld is factual or that it is complete: these requirements can only be enforced by the humans who supply the information to the system.

Being a set, the UoD is *fixed*: it does not change with time; it does not have states. We also regard the task subworld as fixed. However, what is asserted of this subworld typically does change with time. Informally, we may regard the information system to include a "varset" (set variable) of sentences known as the **knowledge base** (KB). At any time, the sentences in KB are taken to assert propositions about the task subworld. Ideally, these

propositions will be true, and in this sense the KB records the system's explicit "knowledge" about the task subworld. As explained in the next section, further knowledge may be inferred by applying logical inference to the rules and facts stored in KB.

The knowledge base consists of a (fixed) set of sentences known as the **conceptual schema** (CS) and a varset of sentences known as the **database** (DB). The CS delimits the UoD, and the database viewed across time determines which of the UoD subworlds is chosen to model the application. While practical systems sometimes permit the CS (and hence UoD) to evolve, along with the user's perception of what ought to be the task, for the purposes of this thesis we assume the CS and UoD are stable.

The knowledge base then is the union of the conceptual schema and the database. While this definition is also used by some other researchers, the term "knowledge base" is sometimes used in other ways (e.g. some researchers exclude the database from being part of the knowledge base).

Typically, each sentence in the database is an *elementary fact*: roughly, this is an instantiation of an irreducible logical predicate, e.g. "the Person with surname 'Halpin' seeks the Degree with code 'PhD'". During the lifetime of the information system, one particular UoD subworld (ideally one matching the task subworld) is described by the system. However, from the point of view of humans interacting with the information system, only some of this description may be available at any given time.

At any time, the database expresses the current elementary assertions about the application. Besides current events, these assertions may concern past events (e.g. sales figures for last two years) and (anticipated) future events (e.g. airline flight timetable for next week). The database may grow or shrink when an update occurs. An update may add facts, delete facts or both. A simplified picture of an evolving knowledge base is provided by Figure 3.2. Here the vertical dimension measures the total number of sentences stored in the KB. The CS component is stable but the DB component goes through a series of changes as it is updated (typically through compound transactions).

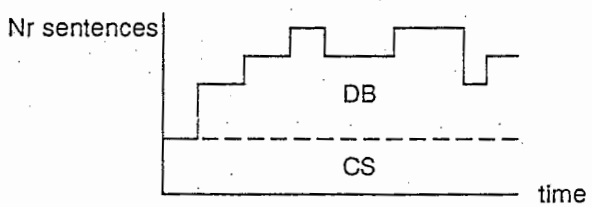


Figure 3.2 Evolution of the knowledge base

The truth value of a proposition does not change with time. Assuming invariant reference, some sentences express the same proposition regardless of their time of utterance. These include sentences expressing analytic propositions (e.g. "A square has 4 sides"), and event descriptions which explicitly state the time of the event (e.g. "Einstein was born in 1879"). Time-dependent sentences may express different propositions depending on their time of utterance, e.g. "Reagan is president of the USA" uttered in 1988 expressed a true proposition, but if uttered now would express a false proposition.

When a contingent, time-dependent, present-tense/past-tense sentence *s* is asserted at time *t* (e.g. when added to the database) we take it that this sentence is short for: "At/(At some time before) *t* it is true that *s*". With this understanding, database states may be indexed to their time, and the atemporal UoD subworld cannot result in contradictory propositions (e.g. "Reagan is president of the USA at time t1 " does not contradict its replacement "Bush is president of the USA at time t2").

The CS is intended to precisely define the UoD by specifying what counts as a UoD subworld. Any subworld is determined by the set of relevant atomic propositions true of it. Given a standard interpretation of words, and time indexing of sentences, the UoD subworld described in an information system is determined by a knowledge-time worm (for a rough two-dimensional picture of such a worm, consider the region under the graph in Figure 3.2). Different knowledge-time worms describe different subworlds.

Hence for a given UoD, the CS specifies not only what database states are possible, but also what transitions between database states are possible. For example, the transition from a marital status of "divorced" to "single" might be ruled out. However, while we have set up our framework to include this dynamic aspect, for this thesis *we ignore transition constraints*. For many applications, transition constraints are rare, and can be trivially added and implemented; in some cases these can be expressed by means of behavioural aspects (triggers).

In cases where the role of time is central, or where previous information states are to be preserved, more complex approaches may indeed be required (e.g. temporal logic for conservative databases). A wealth of research literature on the dynamic aspects of knowledge bases exists, and a serious study of these matters is beyond the scope of this thesis.

3.2 Conceptual architecture of an information system

Conceptually, we take the architecture of an information system to be as portrayed in Figure 3.3. Arrow heads indicate the direction(s) of information flow. All communication between the user(s) and the system is via the *conceptual information processor* (CIP). For a given application, users enter the specific conceptual schema, update the database, and request information. Hence the CIP has three functions: CS filter; update filter; information supplier.

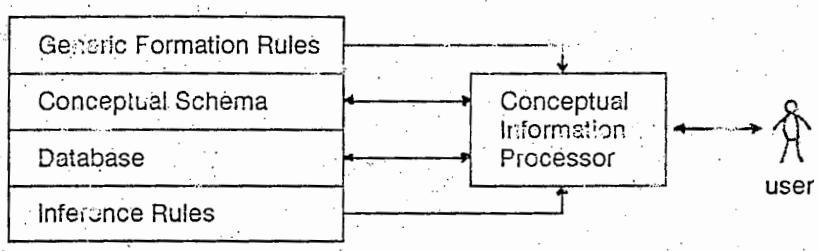


Figure 3.3 Components of an information system (plus user)

To be accepted, user messages must conform to the formal information system language (ISL) used for all applications. This is composed of a generic conceptual schema language (CSL), a generic database update language (UL), and a generic query language (GQL). UL includes the rules for asserting facts (FAL). We focus our attention on the generic knowledge base language KL, which is simply CSL ∪ FAL. The formation rules of KL are detailed later.

The reader who is unclear about the meaning of terms such as "formation rules", "axioms", "inference rules", "proof theory", "model theory", "metalanguage", "consistency", "soundness", "completeness", "decidability", "independence", "isomorphism" etc. should consult Appendix I, which provides relevant background concerning the nature and purpose of formalization.

The following components of the information system are the same for all applications: generic formation rules of ISL; generic CS axioms; inference rules. The CS consists of generic axioms and specific axioms. The *generic CS axioms* include the relevant axioms of logic, mathematics and NIAM. We discuss these later. The *specific CS axioms* are divided into three groups: stored fact type declarations; population constraints; and derivation rules. Among other things, the fact type declarations restrict the symbolic names (e.g. predicate names). When the specific CS axioms are entered by the designer the CIP uses these restrictions together with the CS formation rules to accept or reject the input.

Once the CS is accepted, it is used (with the UL rules) by the CIP to determine whether an update request from the user is to be accepted.

When the user issues a well formed request for information, the CIP searches the knowledge base for the answer. If the relevant information is not stored, the CIP applies its inference rules in an attempt to derive the required information. For example, using Modus Ponens and Universal Instantiation as inference rules, the CIP may deduce that Pat is cancer-prone from the fact that Pat smokes and the derivation rule that if an object smokes it is cancer prone.

The same proposition may be expressed by different sentences. Given that all KB sentences must conform to KL, this narrows down the possibilities. However, different designers may choose vocabularies which differ in their CSL names. For example, designers may choose different symbols for the same predicate (e.g. "is_employed_by" instead of "works_for") or may choose to perceive certain aspects of the UoD in terms of predicates and objects instead of just predicates (e.g. "has_gendercode 'm'" instead of "is_male"). We return to this issue later when defining implication and equivalence between different conceptual schemas.

A formal system consists of a formal language together with a deductive apparatus composed of axioms and/or inference rules. As our interest is in information structures, the general formal system we wish to discuss is as set out in Figure 3.4. We call this "KS" (Knowledge System).

Language:	KL
Axioms:	generic CS axioms
Inference rules:	MP, etc.

Figure 3.4 The general formal system KS

For a given application, a specific formal system is obtained by adding specific CS axioms for that UoD (see Figure 3.5). Different designers might devise different specific formal systems S1, S2 etc. for the same application. Such systems can differ only in the specific CS axioms.

Language:	KL
Axioms:	generic CS axioms
	specific CS axioms
Inference rules:	MP, etc.

Figure 3.5 A specific formal system

One might consider a further level of specific formal systems by adding the facts expressed in a database state as axioms. However, we are primarily interested in examining the connections between two different conceptual schemas, say CS1 and CS2. Such connections can be explored proof-theoretically, e.g. within the general system KS are the specific axioms of CS2 deducible from those of CS1? These connections can also be explored from the viewpoint of model theory, e.g. is every model of CS1 also a model of CS2? Intuitively, knowledge-time worms are closely related to models.

Two major concerns for this chapter and the next three chapters are to spell out the details of the KS system, and to capture our model-theoretic intuitions within a formal framework. Appendix I includes some general reasons as to why formalization is of importance. Within this thesis, the main motivation for our formalization is to provide a rigorous framework for establishing whether two conceptual schemas are equivalent.

3.3 The formal language QL=

In this section and the next we discuss the syntax and formal semantics of our knowledge base language KL, which is based on first order predicate calculus with identity and functions, but tailored for knowledge base work. Our general approach to predicate calculus is set out in Halpin & Girle (1981); further background on identity, functions and metalogic are provided by Rennie and Girle (1973) and Hunter (1971).

Formalization of knowledge bases in terms of first order logic has been done before (e.g. ISO 1982 Appendix F), but we believe our approach provides a more natural framework, with clear links to NIAM concepts. The only other work we are aware of which seeks to provide some "formal" basis for NIAM uses the NIAM language itself (e.g. Leung 1988) or RIDL (Meersman, 1981) or Prolog (McGrath, 1987). To our knowledge, our formalization is the first which provides a rigorous model theory and proof theory for NIAM.

First we summarize *QL*= (Quantification Language with identity), a basic predicate calculus language. QL= is like an assembly language: it is powerful enough to express all our knowledge base requirements, but is often awkward to work with. We later introduce notational variations and define higher level constructs until we arrive at a language which facilitates the textual translation of NIAM diagrams.

Though spartan, QL= is not parsimonious, e.g. ~ and & would suffice for the propositional operators, and functions could be expressed as many:one predicates. Propositional variables are excluded since there is no need for them. The propositional and individual constants are used only in formal proofs. In the following syntactic definitions, a sequence of one or more space characters is used as a metasymbol for "or" (instead of "|") and nonterminals are italicized.

propositional constant:	T F	
individual constant (IC):	a b c d e a_0 a_1 a_2 ..	
individual variable (IV):	x y z w v u x_0 x_1 x_2 ..	
function variable (FV):	f^n g^n h^n f_0^n f_1^n ..	(where arity $n \geq 1$)
function constant (FC):		(no FCs in QL=)
function symbol:	*FV FC*	
predicate variable (PV):	F^n G^n H^n F_0^n F_1^n ..	(where arity $n \geq 1$)
predicate constant (PC):	$=^2$	
predicate symbol:	*PV PC*	
quantifier:	∀ ∃	
parenthesis:	( )	
propositional operator:	~ & ∨ → ≡	

Propositional, predicate and function constants have special axioms or rules which apply in all interpretations. In QL=, only three such constants (T, F, $=^2$) are used: we explain these shortly. *Terms* and *wffs* (well formed formulas) of QL= are defined as follows.

Term formation rules:

Basis clause:	Each IC and IV is a term.
Recursive clause:	If *f* is an n-ary function symbol and $t_1..t_n$ are terms (not necessarily distinct) then $f(t_1,..,t_n)$ is a term.
Terminal clause:	If *t* is a term, it is so because of the above rules.

Wff formation rules:

Basis clauses:	T and F are wffs
	If R is an n-ary predicate symbol and $t_1..t_n$ are terms (not necessarily distinct) then $Rt_1..t_n$ is a wff
Recursive clauses:	If α is a wff, so is $\sim\alpha$
	If α is a wff and *v* is an IV, then $\forall v\alpha$ is a wff
	If α is a wff and *v* is an IV, then $\exists v\alpha$ is a wff
	If α and ß are wffs, so is $(\alpha \,\&\, ß)$
	If α and ß are wffs, so is $(\alpha \vee ß)$
	If α and ß are wffs, so is $(\alpha \rightarrow ß)$
	If α and ß are wffs, so is $(\alpha \equiv ß)$
Terminal clause:	If α is a wff, it is so because of the above rules.

For our purposes we restrict wffs further to *closed wffs*, i.e. wffs in which each IV is bound to a quantifier. Closed wffs are said to be *sentences*. The standard *semantics* of a set of QL= sentences is now summarized. A *tuple* is a sequence of items, and a *relation* is a set of tuples. An n-ary *function* is an n+1-ary relation in which there is only one tuple with the same first n items: these n items are said to be the *arguments* of the function and the n+1th item is the *value* of the function.

An *interpretation* I of a set of QL= sentences comprises a non-empty domain D of *individuals* (each of which is named by a constant), together with the following assignments:

- The propositional constants "T" and "F" are respectively assigned the truth values True and False.
- Each IC is mapped onto one individual in D.
- Each predicate symbol is mapped to a relation over D.
- The predicate constant "$=^2$" is interpreted as the identity relation.
- Each function symbol is mapped to a function with arguments and values in D.
- The operators ~, &, V, → and ≡ are given their usual truth-functional interpretations (negation, conjunction, inclusive disjunction, material implication and material equivalence respectively).
- quantifiers are interpreted by expansion on D, e.g.
  $\exists x\, F^1x \equiv F^1a \vee F^1b \vee ..$
  $\forall x\, F^1x \equiv F^1a \,\&\, F^1b \,\&\, ..$

An interpretation I of QL= is a *model* of a set of QL= sentences iff each sentence in this set is true for I. For a more detailed treatment of these standard semantics see Hunter (1971 pp. 141-9).

A formal system comprises a formal language and a deductive apparatus. A deductive apparatus for QL= may be provided in many ways. For example, the propositional part may be axiomatized by the three axiom schemata $(\alpha\rightarrow(\beta\rightarrow\alpha))$; $((\alpha\rightarrow(\beta\rightarrow\gamma))\rightarrow((\alpha\rightarrow\beta)\rightarrow(\alpha\rightarrow\gamma)))$; $(\sim\sim\alpha\rightarrow\alpha)$ together with definitions for &, V and ≡ in terms of → and ~, and the inference rule AA: If α and $(\alpha\rightarrow\beta)$ are theorems so is β. The AA rule (Affirming the Antecedent) is often called MP (Modus Ponens). The predicate component without identity is usually axiomatized with at most four axiom schemata (e.g. see Hunter 1971 p. 167), and the identity relation may be axiomatized by two rules (reflexivity: $\forall x =^2 xx$) and substitutivity of identicals (SI). SI says that if $=^2xy$, and $\Phi(y/x)$ is like Φx except for having free occurrences of y in zero or more places where x occurs free in Φx, then each closure of $\Phi x \rightarrow \Phi(y/x)$ is a theorem (e.g. see Hunter 1971 pp. 196-7).

However, instead of the axiomatic approach, we set out "proofs" in QL= by using a technique we call *deduction trees*. Deduction trees are essentially semantic tableaux, with the added freedom to evaluate branches more swiftly by using natural deduction at any stage. Our basic treatment of semantic tableaux is set out in Halpin & Girle (1981). For the identity relation we add SI (substitutivity of identicals) and the rule that any branch with a node of the form $\sim=^2t_1t_2$ may be closed where t_1 and t_2 are ground terms (e.g. see Rennie & Girle 1973, pp. 187-8). A *ground term* is an IC or a function term whose arguments are ICs or ground terms, e.g. "a", "$f^1(a)$", "$g(a,f^1(b))$". Ground terms are also called closed terms.

While our proofs and counterexample generation are based upon the predicate calculus for QL=, knowledge bases must conform to additional axioms and formation rules, as discussed later.

In principle, each knowledge base state can be expressed as a set of sentences of QL=. Informally, an interpretation of a KB state corresponds to an atemporal glimpse of a possible world, with the domain of the interpretation being the set of objects (individuals) perceived in that glimpse. Notice however, that QL= is untyped. Each individual constant or ground function term denotes one individual. Different ICs and different ground function terms may all refer to the same individual. This situation may be portrayed by an occurrence diagram as in Figure 3.6.

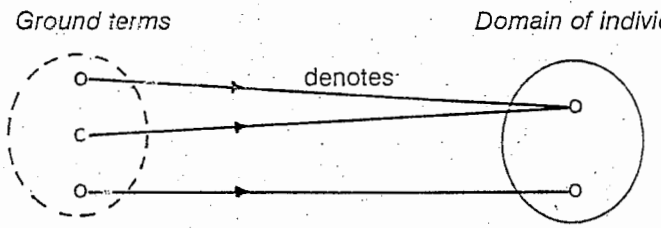


Figure 3.6 Each IC or ground function term denotes one individual

The untyped, simplistic reference scheme of QL= is awkward for humans since it ignores the natural perceptual tendency of humans to categorize objects into types (e.g. Person, Department), and its names are artificial and forgettable (e.g. "F^2ab" might be used for "The person with name 'Halpin TA' works for the department with title 'Computer Science'"). In the next section we modify the QL= language to include a richer range of identifiers and make it notationally more readable. We also show how types can be emulated by predicates, and define certain predicate and function constants to be included in all knowledge bases. The language so formed we call KL.

3.4 The formal language KL

Our knowledge base language KL is built on top of QL=. To begin with, we adopt the following rules to aid readability of formulas. The new constructions are well formed and have the semantics of their definiens.

1. The arity of a predicate or function symbol is implicitly specified in context by the number of its arguments. For example, F^2xy may be written as Fxy. In principle, if not in practice, we allow wffs such as $\forall x\exists y(Fx\&Fxy)$ since this is short for $\forall x\exists y(F^1x\&F^2xy)$ which has two distinct predicates F^1 and F^2.

2. Square brackets may be used like parentheses to delimit wffs, i.e.
   If (α) is a wff then so is $[\alpha]$

3. Outermost brackets may be dropped from a formula occurrence which is whole (not a proper subformula), i.e.
   If (α) or $[\alpha]$ is a whole wff then so is α

4. Brackets may be dropped in accordance with the following priority convention for propositional operators, with sequences of the same operator evaluated left-associatively.

   highest ↓ lowest

$\sim$	(already implied by formation rules)
$\&$	
$\vee$	
$\rightarrow$	
$\equiv$	

   e.g. $Fx \vee Gx \,\&\, Hx =_{df} Fx \vee (Gx \,\&\, Hx)$
   $Fx \vee Gx \vee Hx =_{df} (Fx \vee Gx) \vee Hx$

   Note that =, being a predicate, has higher priority still,
   e.g. ~x=y is equivalent to ~(x=y) { here = is shown infix: see later }

5. Extra brackets around wffs are allowed, i.e.
   If α is a wff, so are (α) and $[\alpha]$

6. If $v_1..v_n$ are IVs, and Q is uniformly $\forall$ or $\exists$
   then if α begins with (, [, $\forall$, $\exists$ or ~
   then $Qv_1..v_n\alpha =_{df} Qv_1..Qv_n\alpha$
   else $Qv_1..v_n\alpha =_{df} Qv_1..Qv_n\alpha$ { here "␣" denotes " " }

   e.g. $\forall xy(Fxy\rightarrow Gxy) =_{df} \forall x\forall y(Fxy\rightarrow Gxy)$
   $\forall xy\exists zwv\, Fxyzwv =_{df} \forall x\forall y\exists z\exists w\exists v Fxyzwv$

   Note that formulae like $\exists yxFy$ and $\forall xxFx$ are not well formed.

We do not expand the set of identifiers for individual variables, but we do allow more meaningful identifiers to be used for function and predicate symbols. As a preliminary, the EBNF syntax of some relevant syntax groups is now set out. Here, a space character " " is used as a metasymbol for "or". Nonterminals are italicized. [x] means x is optional. {x} means 0 or more occurrences of x. Familiar sequences are abbreviated with the use of "..".

lowercase letter (*ll*):	a b .. z
uppercase letter (*ul*):	A B .. Z
letter (*l*):	*ll ul*
digit (*d*):	0 1 2 3 4 5 6 7 8 9
identifier_char (*idch*):	*ll ul d* _ # $ %

7. function_id: *ll*{*idch*}

This production rule expands rather than replaces the previous syntax. Some newly permitted function terms are: cube_of(a); sale(a,b).

The inclusion of new predicate identifiers is somewhat more complex. To begin with we add the rule:

8. prefix_predicate_id: *ul ul*{*idch*}␣

If R is a prefix_predicate_id and $t_1,...,t_n$ are terms, then $Rt_1..t_n$ is a wff.

Here "␣" denotes the space character " ". Any predicate symbol with more than one unsubscripted character must be terminated by a space. Some newly permitted wffs are: Person x; Likes ab

So far all predicate symbols are written in prefix position. We now allow binary predicate symbols to be written *infix*. Here "..." is a place-holder ellipsis, and has a separate usage from "..". Unless the symbol is an upper-case letter or "=", it must be a lower-case letter followed by at least one character and flanked by spaces.

9. infix_predicate_id: *ul* ...␣*llidch*{*idch*}␣... =

If R is an infix_predicate_id with no "..." and t_1, t_2 are terms then t_1Rt_2 is a wff.
If ...R... is an infix_predicate_id and t_1, t_2 are terms then t_1Rt_2 is a wff.

Some newly permitted wffs are: xRy; x=y; a works_for b; but not a x b.
In the third case the predicate symbol is "... works_for ...".

We allow predicates of arity above 2 to be written in *mixfix* (or distfix) form. These are italicized, with spaces separating them from their terms.

italic_identifier_ch (iic): italic version of *idch*.

10. mixfix_predicate_id: ..._*iic*{*iic*}_..._*iic*{*iic*}_...{_*iic*_...}

    If *R* is a mixfix_predicate_id then the result of substituting each "..." in *R* with a term is a wff.

    e.g. "x *scores* y *for* z" uses the predicate "... *scores* ... *for* ...".

Note that mixfix predicates are terminated by place-holders, e.g. "A score of .. was obtained by .. for .." is illegal. The language could be extended further to permit this and allow postfix predicates (e.g. "... is_male"), but we do not include such extensions within this thesis.

So far, all spaces have served as delimiters in our syntax. We also permit extra spaces around delimiters as follows:

11. Extra spaces are allowed around propositional operators, =, quantifiers, brackets and spaces.
    e.g. Fx ∨ Gx; x = y; ∀x Eyz (Fx & yGz)

We also wish to extend the range of individual constants. Before doing so it will be helpful to classify the individuals in the universe of discourse. Recall that any knowledge base state may be given an interpretation which includes specification of the domain of individuals D. Now from the human perspective it is natural to *partition* this domain into a set of mutually exclusive and collectively exhaustive populations. Figure 3.7 illustrates a partition which applies to all our knowledge base interpretations.

The domain is partitioned into five classes. For each of these classes, some sample members have been depicted by constant terms of either a textual or graphic nature. We briefly describe each of these classes informally to convey the main ideas, then provide a formal treatment.

Strings are abstract symbols but are denoted in a direct way by those marks which are spatial sequences of characters: naively, strings may be "written"; they are *lexical entities*. The string 'Ann' is the 3-character spatial sequence inside the quotes. All other entities are nonlexical.

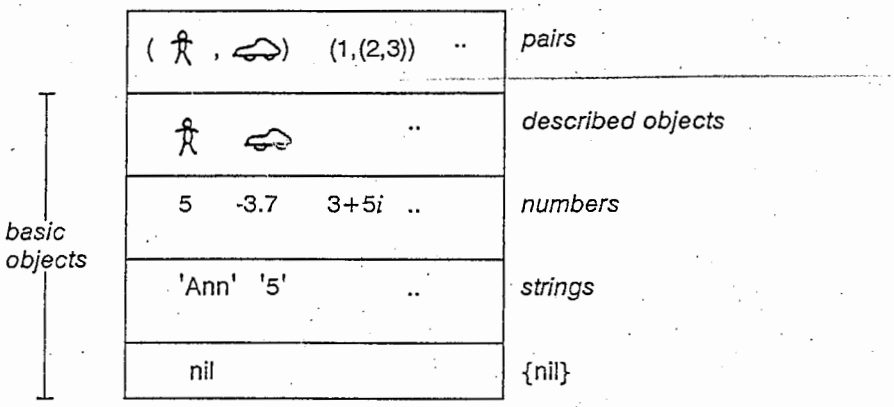


Figure 3.7 The domain of objects is partitioned into 5 classes

Numbers are abstract entities for which we define various mathematical operations. We represent them by numerals (i.e. strings which conform to the syntax of numeric terms). Sometimes we may wish to pose queries about strings or numbers themselves (e.g. Which surname is also a name of a city? List all subject codes starting with "CS". Whose mass in kg is numerically greater than his/her IQ?). Moreover, the explicit specification of reference schemes for *described objects* (e.g. People, Cars, Lengths) involves strings or numbers, and derivation rules often do likewise. In Figure 3.7, we have depicted two described objects (a particular person and car) by means of pictures. However, described objects are always referenced in the KB by means of *definite descriptions*, e.g. "the person with surname 'Halpin'". The other objects are referenced in the KB by specialized ICs or function terms of KL.

We use a typeless first order formalization in which predicates and functions legally range over the whole domain. Moreover, we give ground function terms the same ontological status as individual constants, viz. any such term does refer to some object in the domain. We include the *nil* object in our domain as the referent for all simple "garbage" expressions, such as "2+'red'". We may pick any concrete or abstract object as nil, so long as it isn't a string, a number or described, e.g. we might pick my current garbage bin as nil! The nil object is always denoted by "nil", which we now add as an IC to KL. If desired, nil may also be treated as the empty list, and used to construct lists in the usual way.

Objects in the four classes so far discussed are called *basic objects*. The final class of objects consists of objects which are *ordered pairs*. The pairing operation may be applied to any two objects (basic objects or pairs), and always produces a pair. So the domain is closed under pairing. Later in this section we axiomatize the pairing operation, using special function terms

to denote pairs. Our motivation in including pairs is to simplify later work with nested fact types. Apart from including described objects, our domain partitioning is quite similar to the scheme used in the language Trilogy (Andrews 1987, p. 32).

Note that our ontology is not parsimonious. From an abstract point of view, any basic object could have been chosen as the nil object, and both multi-character strings and multi-dimensional numbers could have been constructed as special sequences using the pairing operation. However, we feel our classification scheme is closer to the way in which such objects are perceived in practice. For example, a two-character string is typically thought of as a spatial ordering of characters, and we might wish to consider the pair (3,9) without the connotation of a complex number.

In order to express the partition in Figure 3.7, we treat the following prefix predicate identifiers as predicate constants with the fixed interpretation shown in braces:

Basic x	{ x is a basic object }
Described x	{ x is (definitely) described by the user }
Number x	{ x is a number }
Pair x	{ x is a pair }
String x	{ x is a character string }

The following *partition axioms* are included in our knowledge base system KS. When first presented, the names of KS axioms are displayed in bold.

P1 $\forall x[\ (\text{Basic } x \vee \text{Pair } x)\ \&$
$(\text{Basic } x \equiv \text{Described } x \vee \text{Number } x \vee \text{String } x \vee x = \text{nil})\ ]$

P2 $\forall x[\ \sim(\text{Basic } x\ \&\ \text{Pair } x)\ \&$
$\sim(\text{Described } x\ \&\ \text{Number } x)\ \&$
$\sim(\text{Described } x\ \&\ \text{String } x)\ \&\ \sim(\text{Number } x\ \&\ \text{String } x)\ ]$
$\&\ \sim \text{Described nil}\ \&\ \sim \text{Number nil}\ \&\ \sim \text{String nil}$

By giving "nil" the status of an IC, it follows that nil exists in every domain, i.e. $\exists x\ x=\text{nil}$ (this may be trivially shown with a deduction tree). Later we introduce ground terms for real numbers and strings; so these also exist in all domains. We see later that if two objects exist, so does their pair. Details about the described entities (if any) are completely dependent on the particular KB. However, we provide within KS a minimal semantics for

Number, String and Pair. Let's start with Number. Although we included this general category to enable the set of n-dimensional numbers to be treated as a subset of the set of $n+1$-dimensional numbers, for $n \geq 1$, in this thesis we focus our attention on the *real numbers*. We begin by adding the following predicate and axiom:

Real x { x is a real number }

RN $\forall x$(Real x → Number x)

We now expand the symbols of KL to enable numbers to be referenced in the usual notation, and add a first order set of axioms for the closed field of real numbers, relativized to individuals instantiating the Real predicate.

To begin with, we add to KL the individual constants 0 and 1, two unary function constants - and $^{-1}$, and four binary function constants +, −, * and /. For convenience, we write unary - as a prefix operator, $^{-1}$ as a postfix operator, and $+^2$, $-^2$, $*^2$, $/^2$ as infix operators using the priority convention

$$\downarrow \quad \begin{array}{ll} - \ {}^{-1} & \{ \text{unary} \} \\ * \ / & \\ + \ - & \end{array}$$

with operators on the same level evaluated left-associatively, and we allow extra spaces around these operators, e.g.

$$x + y * -z / w^{-1} \quad =_{df} \quad +(x,/(*(y,-(z)),{}^{-1}(w))$$

We also add "≠" to KL as a derived symbol for inequality, with the following definition (where x and y are any terms):

$$x \neq y \quad =_{df} \quad \sim(x = y)$$

We now set out the ten field axioms, relativized to reals and including closure where needed. Given our left associativity convention, the left operand of = in RF2 and RF6 could be written without parentheses.

RF1 $\forall xy$ [Real x & Real y → Real x+y & x+y = y+x]
RF2 $\forall xyz$ [Real x & Real y & Real z → (x+y)+z = x+(y+z)]
RF3 Real 0 & $\forall x$[Real x → x+0 = x]
RF4 $\forall x$ [Real x → Real -x & x + -x = 0]

RF5 $\forall xy$ [Real x & Real y → Real x*y & x*y = y*x]
RF6 $\forall xyz$ [Real x & Real y & Real z → (x*y)*z = x*(y*z)]
RF7 Real 1 & $\forall x$[Real x → x*1 = x]
RF8 $\forall xyz$ [Real x & Real y & Real z → x*(y+z) = (x*y)+(x*z)]
RF9 $1 \neq 0$
RF10 $\forall x$ [Real x & $x \neq 0$ → Real x^{-1} & $x*x^{-1} = 1$]

When applied to real numbers, unary - and $^{-1}$ give the additive and multiplicative inverses, while + and * give the sum and product. Subtraction and division are now defined in terms of these operations:

$$x - y =_{df} x + -y$$
$$x / y =_{df} x * y^{-1}$$

We now add ≤ as an infix predicate constant, and axioms to assert that it is transitive and antisymmetric, and it provides a total order for the reals:

TO1 $\forall xyz$ (x ≤ y & y ≤ z → x ≤ z)
TO2 $\forall xy$ (x ≤ y & y ≤ x → x = y)
RTO3 $\forall xy$ (Real x & Real y → x ≤ y ∨ y ≤ x)

We now add two axioms to ensure that the reals form an ordered field:

ROF1 $\forall xyz$ [Real x & Real y & Real z & x ≤ y → x+z ≤ y+z]
ROF2 $\forall xyz$ [Real x & Real y & Real z & x ≤ y & 0 ≤ z → x*z ≤ y*z]

We now add the following abbreviations (x and y are any terms):

$$x \geq y =_{df} y \leq x$$
$$x < y =_{df} x \leq y \,\&\, x \neq y$$
$$x > y =_{df} y < x$$
$$x^n =_{df} x * x * .. * x \quad \text{(where there are } n \text{ occurrences of } x\text{)}$$

To complete this first order theory of real closed fields we add one axiom and two axiom schemata. The first axiom (RC1) ensures that cardinal numbers have real square roots:

RC1 $\forall x$ [Real x & 0 ≤ x → $\exists y$(Real y & $x = y^2$)]

RC2 says that every real polynomial of odd degree has a real zero, and RC3 says that 0 is not a sum of nontrivial real squares:

RC2 $\forall y_0..y_n[\ \text{Real } y_0 \ \&\ ..\ \&\ \text{Real } y_n \rightarrow \exists x(\ \text{Real } x\ \&\ y_0 + y_1*x + .. + y_n*x^n = 0\ )\ ]$ for $n = 1,3,5,..$

RC3 $\forall x_0..x_n[\ \text{Real } x_0\ \&\ ..\ \&\ \text{Real } x_n\ \&\ x_0^2 + .. + x_n^2 = 0 \rightarrow x_0 = 0\ \&\ ..\ \&\ x_n = 0\ ]$ for $n = 0,1,2,..$

The first order theory of real closed fields was shown to be decidable by Tarski (1949). It is however at least exponentially complex. Further background on this topic is given by Rabin (ed. Barwise 1977, sec. C.3), and Chang & Keisler (1977 sec. 1.4).

Constants for rational numbers other than 0 and 1 are introduced as abbreviations for functional terms in the usual way, e.g.

$$2 =_{df} 1+1, \quad 3 =_{df} 1+1+1, .. \qquad 0.1 =_{df} 1/10, \quad 0.2 =_{df} 2/10, ..$$

So any ground term of the form $[-]d\{d\}.d\{d\}$ is now assigned a specific real number in all interpretations. Such terms are called, slightly misleadingly, *numeric constants*. Irrationals may either be referenced by description (e.g. Real x & $x^2 = 2$) or be approximated by a rational (e.g. 1.414). Since the system knows that 0 and 1 are reals, and that reals are closed under the operations discussed, it knows that Real 2, Real 3 etc. Moreover, the system can deduce facts such as 170 > 150 and 170 = 2 * 85 from its axioms. As well, relevant subsets of the reals may now be defined. The following subsets are important enough to include in KS.

Integer x	{ x is an integer, i.e. .. -2 -1 0 1 2 .. }
Cardinal x	{ x is a cardinal number, i.e. 0 1 2 .. }
Posint x	{ x is a positive integer, i.e. 1 2 .. }

We fix the interpretation of these predicates by the following axioms. Note that in this thesis we do not employ different notations to distinguish definitions (recursive or otherwise) from the other axioms

RS1 $\forall x[\ \text{Integer } x \equiv x = 0 \vee \exists y(\text{Integer } y\ \&\ (x = y+1 \vee x = y-1))]$

RS2 $\forall x(\ \text{Cardinal } x \equiv \text{Integer } x\ \&\ x \geq 0\ )$

RS3 $\forall x(\ \text{Posint } x \equiv \text{Integer } x\ \&\ x > 0\ )$

We now expand the set of ICs in KL and add axioms to KS to cater for those objects which are character *strings*. Intuitively, strings are sequences of 0 or more characters. We start by adding ICs known as *character constants* for each of the characters introduced earlier (letters, digits, operators, etc.). We adopt the Modula convention. If *c* is a character other than the double-quote (") then the 3-character symbol "*c*" denotes *c*. The double-quote is denoted by the 3-character symbol: '"'. Our first two string axioms are now set out.

Char x { x is a character of KL }

ST1 ∀x(Char x ≡ x="a" ∨ x="b" ∨ .. ∨ x="'" ∨ x='"')
ST2 ∀x(Char x → String x)

Since the list of characters is finite and given, axiom ST1 may be unabbreviated by the patient reader. Though not needed, we allow single-quoted character constants for characters other than single-quote, i.e. '*c*' $=_{df}$ "*c*" when *c* is not the single-quote ('): we often use single quotes when the context is a formula line rather than a paragraph. We use two contiguous single-quotes ('') as an IC denoting the *null string*: intuitively, this is a sequence of 0 characters (its existence might not be very intuitive, but the same could be said of 0). Two double-quotes ("") may be also be used to denote the null string, but we avoid this to save confusion with the constant for double-quote ('"').

We now allow the + operator to be used for concatenation of strings. The following axioms assert that *string concatenation* is commutative and associative, and define the null string.

ST3 ∀xy [String x & String y → String x+y & x+y = y+x]
ST4 ∀xyz [String x & String y & String z → (x+y)+z = x+(y+z)]
ST5 String '' & ∀x [String x → x+'' = x]

Since + is left-associative, the brackets in the left operand of = in ST4 could have been dropped. We now allow string concatenation terms to be abbreviated as follows. If "c_1", .., "c_n" are *n* character constants then "$c_1..c_n$" $=_{df}$ "c_1"+..+"c_n", for $n \geq 2$. For example: "abc" $=_{df}$ "a"+"b"+"c". Similarly, '$c_1..c_n$' $=_{df}$ 'c_1'+..+'c_n', for $n \geq 2$. For example: '"Hi"' $=_{df}$ '"'+'H'+'i'+'"' $=_{df}$ '"'+"H"+"i"+'"'. No abbreviations are defined for terms with mixed quotes, e.g. '"'+'"': in practice such terms are not used.

Hence each string is given a name, even though some of these are just abbreviations for function terms (cf. numeric constants). These names are called *string constants*. Although strings are abstract entities, we may adopt the intuitive view that a string constant is a string which denotes the string that is the character sequence inside its quotes, e.g. the string constants

"CS112" "Don't worry." 'Be "happy".'

denote the strings:

CS112 Don't worry. Be "happy".

The use of a string constant as a term x implies the truth of String x, since the ST axioms identify characters, the null string, and their concatenations as strings. We now define a function *len* which, when applied to a string, returns the *length* (i.e. the number of characters) of that string. The next two axioms enable the length of any string to be computed. The theorems that len('') = 0 and ~Char '' are now trivial to prove. The length of multi-character strings may be computed by expressing them in the form *c+s* where *c* is a character.

len(*s*) { length of string *s* }

ST6 ∀x [Char x → len(x) = 1]

ST7 ∀xy [String x & String y → len(x+y) = len(x) + len(y)]

Axioms TO1 and TO2 established ≤ as a transitive and antisymmetric operator, and definitions were given for < etc. We now use these operators for lexicographic ordering of strings. To do this we first introduce an *ord* function which assigns a unique cardinal number to each character. For this thesis we adopt the ASCII ordering for standard ASCII characters, and assume the reader may supply distinct ordinal values for the rest. This injection from characters to cardinals may be set out as a single axiom ST8 (shown abbreviated).

ord(*c*) { ordinal number for character *c* }

ST8 ord(' ') = 32 & ord('"') = 34 & ord('#') = 35 & ..

It follows that the character constants denote distinct characters. We call this *theorem CC≠*. When first presented, the names of KS theorems that are not axioms are shown in italics.

CC≠ 'a'≠'b' & 'a'≠'c' & .. & "'"≠'"'

This may be trivially proved with a deduction tree, e.g. to show 'a' ≠ 'b': assume 'a' = 'b', use ST8 and the Real axioms to show ord('a') ≠ ord('b'), then use the assumption and SI to deduce ord('a') ≠ ord('a'), which closes. For convenience we now define two more functions (head and rest) with the following meaning when applied to non-null strings:

head(x) { the first character of x }
rest(x) { the rest of x, i.e. all but the head of x }

ST9 ∀xy [Char x & String y → head(x+y) = x & rest(x+y) = y]

For example: head('BSc') = 'B', and rest('BSc') = 'Sc'. Note that a character is a special case of a string. We now add string axioms for ≤:

ST10 ∀x [String x → '' ≤ x]
ST11 ∀xy [Char x & Char y → (x ≤ y ≡ ord(x) ≤ ord(y))]
ST12 ∀xy [String x & String y & x ≠ '' & y ≠ '' →
(x ≤ y ≡ head(x) ≤ head(y) &
(head(x) = head(y) → rest(x) ≤ rest(y)))]

The axiomatization of ≤, =, <, > and ≥ for strings is now complete. For example, to prove that 'ab' < 'c': assume 'ab' = 'c'; use ST axioms to show 'ab' ≤ 'ab' and ~('c' ≤ 'ab'); use SI to give ~('ab' ≤ 'ab'); closure now gives 'ab' ≠ 'c'; ST axioms give 'ab' ≤ 'c'; combining we have 'ab' < 'c' by definition. To shorten such proofs, the following theorem concerning equality of non-null strings may be proved using TO2 and ST12:

ST= ∀xy [String x & String y & x ≠ '' & y ≠ '' →
(x = y ≡ head(x) = head(y) & rest(x) = rest(y))]

String analogues of TO3 and OF1 are also easily proved. The string axioms ST1..ST12 are adequate to define any subtype of String and in conjunction with our Real axioms may be used to define further string operations if required. However, to simplify the specification of string subtypes (e.g. c20, aaddd -- we discuss such cases in the next chapter), it is convenient to define the following lexical subtypes and include them within KS.

Digit x { x is a digit character }
Letter x { x is a letter }
Digits x { x is a string of 1 or more digits }
Letters x { x is a string of 1 or more letters }

ST13 $\forall x$ (Digit x ≡ x='0' ∨ x='1' ∨ .. x='9')
ST14 $\forall x$ (Letter x ≡ x='a' ∨ .. ∨ x='z' ∨ x='A' ∨ .. ∨ x='Z')
ST15 $\forall x$ [Digits x ≡ String x & x≠'' & Digit head(x) &
(rest(x)≠'' → Digits rest(x))]
ST16 $\forall x$ [Letters x ≡ String x & x≠'' & Letter head(x) &
(rest(x)≠'' → Letters rest(x))]

Elsewhere (NH89 Ch. 7) we allowed for the possibility of refraining from specifying either numeric or string aspects for a symbol, e.g. consider "35" as a RoomNr. However, for simplicity we assume by default that such "atomic symbols" are really strings; it does not matter conceptually if the string operations are not used, and at the implementation level other data types could be chosen if desired. For example, in the conceptual schema we might specify room# as a string reference mode, but when we later specify the internal schema we might choose a numeric data type such as smallint simply to save storage space or to speed up value comparisons.

Typically, numeric and string predicates and functions are not used explicitly when specifying a database (i.e. the set of stored facts), and are usually not depicted explicitly on a NIAM CS diagram. However they are required to specify lexical subtypes (e.g. <aaddd>), some derivation rules (e.g. computing profit from cost and sale prices), some nongraphic constraints (e.g. deathyr not earlier than birthyr), and various *ad hoc* queries (e.g. list all thesis titles containing the phrase "information systems").

We now axiomatize *ordered pairs*. To begin with, we add the following function constant and axiom.

pair(x,y) { the ordered pair comprising x followed by y }

OP1 $\forall x_1 x_2 y_1 y_2 [\ pair(x_1,y_1) = pair(x_2,y_2) \equiv x_1=x_2 \ \&\ y_1=y_2\ ]$

Since ground function terms always refer, if x and y are objects in the domain, so is pair(x,y). For convenience, we allow any term of the form "pair(x,y)", where x and y are terms, to be *abbreviated* to "(x,y)". To prevent confusion with the notation for function argument lists, we stipulate that an occurrence of the form "(x,y)" in a formula may be expanded to

"pair(x,y)" if and only if the occurrence is not immediately preceded by a function symbol. Axiom OP1 may now be expressed more concisely:

$$\forall x_1 x_2 y_1 y_2 [\ (x_1,y_1) = (x_2,y_2) \equiv x_1 = x_2 \ \&\ y_1 = y_2\ ]$$

We make the pairing operation *right-associative*, allowing removal of brackets around embedded pairs. If x, y and z are any terms, and "(x,y,z)" does not follow a function symbol then:

$$(x,y,z) =_{df} (x,(y,z)) \qquad \{ \text{ where each side is a term } \}$$

Since the pair function is recursive, ordered n-tuples of any arity above 1 can be represented as a pair. In this sense, the domain is closed under tupling. We make use of this notation later when we formalize nested fact types.

System ICs (a, b; c, d, e, a_0 ..) are useful in discussing model theoretic matters (e.g. setting out universal and existential instantiations, and depicting a model or countermodel), but are not used for the specification of a conceptual schema or a database. The propositional constants T and F are used only for formal proofs. Any conceptual schema or database may be expressed as a set of closed KL wffs (without system ICs or propositional constants). However, not just any set of wffs will count as a CS or DB. In the next two chapters, we discuss how a knowledge base expressed in the graphical and textual notation of NIAM may be translated into KL.

4 Specifying NIAM conceptual schemas in KL

4.1 Object types and predicates

To clarify the NIAM notation, and to enable theorems about conceptual schemas to be formally proved, this chapter and the next explain how a knowledge base expressed in the NIAM notation may be translated into a set of sentences in KL. Provided formation rules for a NIAM knowledge base can be supplied, this will also determine what counts as a well formed knowledge base in KL. Our formalization also introduces a number of revisions and extensions to NIAM itself (especially in the next chapter).

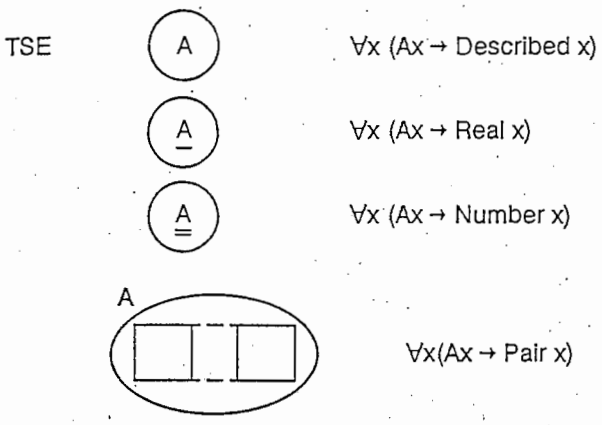
Some of our early ideas on translating NIAM into predicate logic were communicated elsewhere (Nijssen et al. 1988, pp. 3-4, 8-14). However, this chapter provides a much more comprehensive and improved treatment. To begin with, each knowledge base includes the KS axioms discussed previously. We now translate the components of a NIAM CS diagram, starting with the nodes which are *ellipses*. The names of these nodes conform to the syntax of a prefix_predicate_id, and are written inside or beside the ellipse.

Informally, an ellipse may be thought of as enclosing a set of individuals. When the ellipse is *broken* these individuals are *strings* (lexical objects). This is formally captured by translation rule TBE (Translate Broken Ellipse). The translation formula is shown on the right. Translation rule names are shown in plain capitals, starting with "T". We treat TBE etc. as translation schemas, rather than translation instances: here "A" is a predicate variable standing for any unary predicate.

TBE  $\forall x\ (Ax \rightarrow String\ x)$

Solid ellipses may be thought of as enclosing *non-lexical* objects (other than nil). If a solid ellipse encloses role boxes its objects are *pairs*, otherwise its objects are basic. Alternatively, pair types may be depicted by a rectangle surrounding the role boxes. We postpone a detailed discussion of pairs until section 4.6.

A solid ellipse which does not enclose role boxes must be named. If the ellipse name has a single *underline* then the objects are *real numbers*: the underline reminds us of a segment of the real number line. A double underline is used to include higher dimensional numbers: we make no further use of the double underline in this thesis. The rule for translating solid ellipses is set out in TSE.



These underlining conventions are used here for the first time. In our earlier work (NH89) the terms "entity" and "label" usually meant "described object or number or pair" and "string" respectively. In cases where strings played non-referential roles, we spoke of labels that were entities and used a doubly bordered ellipse with both solid and broken lines. We no longer use this notation. Strings are always indicated by a single broken ellipse, and numeric objects are distinguished from described objects by underlining.

Nil is never depicted on a NIAM diagram. Most of what we want to say about an *"object type"* applies for any non-nil object type. So for convenience we introduce the *hybrid ellipse* notation shown in Figure 4.1. An ellipse which is half solid and half broken stands for any non-nil object type. Though the names of such hybrid ellipses are not underlined, numeric object types are included. It should be understood that this is a metanotation only: no hybrid ellipses are used in NIAM diagrams, e.g. the designer must specify whether an object type is lexical.

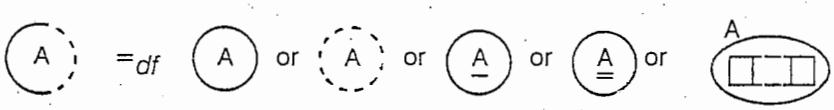


Figure 4.1 Hybrid ellipses are used only in the metatheory

In a NIAM diagram, a *predicate* of arity n is displayed as a named, contiguous sequence of n boxes, where $n \geq 1$, e.g.

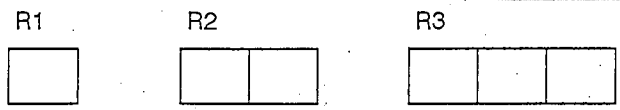


Figure 4.2 Unary, binary and ternary predicates

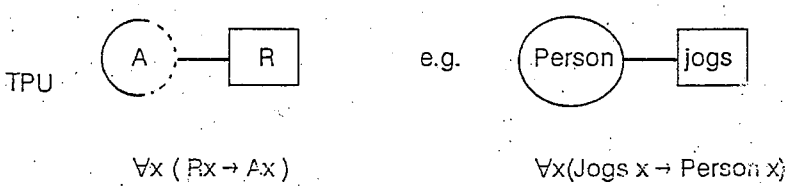
We assume that the designer has chosen to adopt the convention of using a single predicate name (unabbreviated if necessary, as discussed later), written inside or beside the box which corresponds to the first place holder of the predicate, with the spatial order of the boxes (starting at the box with the predicate name) matching the order of the predicate placeholders.

We have discussed elsewhere (NH89, Ch. 3) another way of portraying "fact types" in NIAM, which makes the spatial order of the boxes irrelevant by giving each a role name. If desired, this could be accommodated within our convention by using the ordinal for the placeholder as the (unqualified) role name for the box. For example, see Figure 4.3.

Figure 4.3 The left diagram is equivalent to R2 of Figure 4.2

We regard our ordered predicate notation and the unordered role notation as different only in their focus. In the predicate approach, the predicate name carries the burden of the informal semantics with the role names reduced to ordinals; in the role approach, the role names carry the burden of the informal semantics with the fact type name often reduced to some arbitrary identifier. In both approaches we see a linguistic structure in which objects play various roles with respect to the verb or relationship.

In a NIAM diagram each role box must be connected by an arc to exactly one ellipse. Moreover, each (unabbreviated) predicate name appears only once. Informally, each role is "typed" so that only objects satisfying the predicate of the attached ellipse play that role. Here is the *unary predicate* case:



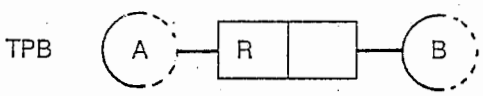
Note that NIAM diagrams, as well as our hybrid ellipses, are topological rather than metric. Ellipses may be shown with any size and oblateness; role boxes may be shown as any rectangle; the contact point of arcs with ellipses and boxes may be anywhere on the perimeter; etc.

We use the example for TPU to make a general point. The NIAM diagram asserts that in the UoD under discussion, no individual jogs unless that individual is a person. There are two basic ways of satisfying this intention when a database is added. Firstly, we could make it (syntactically) *illegal* to enter a fact claiming that some non-person jogs. This would entail adding specific formation rules to the language for each specific predicate.

The second approach is to allow such sentences as wffs but to reject them on the basis that they must be *false*. It is this second approach that we adopt throughout. A specific KB is constructed by starting with KS and adding axioms specific to the application. Once the axiom that ∀x(Jogs x → Person x) is added to KS, any later assertion that a non-person jogs must be rejected since it contradicts this axiom. In this sense then, we simulate "typing" within our untyped calculus.

As a trivial issue, predicate names on a NIAM diagram must be modified if necessary to agree with the syntax of predicate names for KL. With our current version of KL, if we allow NIAM unary predicate names to start with a lower-case letter, this letter should be upshifted (see the TPU example). Also, the predicate constants already specified for KS should be treated as reserved.

The *binary predicate* case is dealt with as follows:



Here, and elsewhere in stating such translations, the object types attached to the predicate under consideration are not necessarily distinct. The right hand example in Figure 4.4 is a case where A = B.

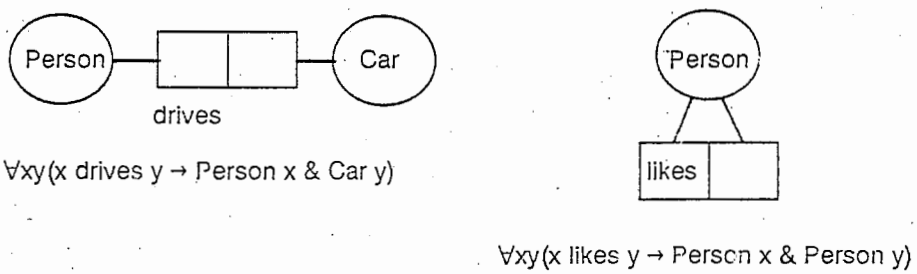
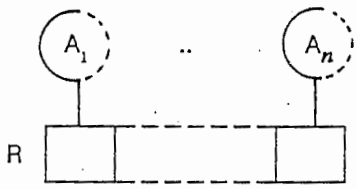


Figure 4.4 Two examples of translating binaries into KL

Ternaries and higher arity predicates are translated similarly. The *n-ary predicate* case is summarized by rule TPN. Here $A_1,..,A_n$ are any non-nil object types, including described types, numeric types, lexical types and pair types (as discussed later, these include subtypes and compositely described object types).

TPN



$\forall x_1..x_n\ (Rx_1..x_n \rightarrow A_1x_1\ \&\ ..\ \&\ A_nx_n)$ for all $n \geq 1$

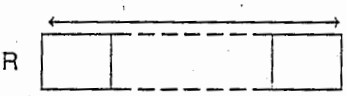
Specifying the fact types for a particular conceptual schema involves two tasks: the names (and arities) of the recognized predicates are listed; and each predicate has its places "typed" as discussed in this section. These declarations may be described as "*typing constraints*", although in NH89 the term "constraint" was used in a more restrictive sense.

Though developed independently, the basic formulation of typing constraints presented here is rather obvious, and has been adopted by other researchers (e.g. in Reiter 1984, formulae of the form TPN are classified as integrity constraints which specify the domains of relations).

4.2 Uniqueness, mandatory role and frequency constraints

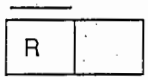
In this section we map three varieties of NIAM graphical constraints into KL, starting with *uniqueness constraints* (UCs). As discussed in NH89 (Ch. 4), each predicate is given at least one UC. The weakest UC is depicted as a bar (or two-headed arrow) spanning the whole predicate. This is really no constraint at all as far as our formalization is concerned, since repetition of a proposition has no logical significance (conjunction is idempotent), and by default any set of tuples of the right type and arity is allowed. Hence such "constraints" are simply ignored (see TUC1).

TUC1



ignore this "constraint"

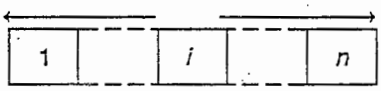
We now consider intra-predicate UCs which span only some of the roles in the predicate. The simplest case of this is catered for by TUC2.

TUC2 

$\forall xyz\ (xRy \ \& \ xRz \rightarrow y = z)$

All our constraint translations assume a *static interpretation*. For instance, in Figure 4.4 adding a UC to the first role of Drives indicates that *for each KB state*, each person drives *at most one* car: this still allows a person to drive several cars during the lifetime of the application.

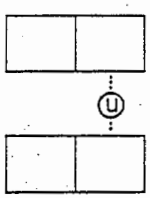
TUC2 is generalized by TUC3 for the case of any UC spanning n-1 roles of a predicate of arity n. Unless the roles are contiguous, arrow tips must be added to the constraint bar (to distinguish a single UC from two separate UCs). To help explain the translation rule, ordinals have been added as role names to indicate that the UC spans all but the *i*th of the n roles. For a fact type of arity n to be elementary, any uniqueness constraint on it must span at least n-1 roles: so TUC1-3 cover all cases of intra-predicate UCs on elementary fact types. We discuss UCs on compound fact types later in this section.

TUC3 

$1 \le i \le n$

$\forall x_1..x_i..x_n y\ (Rx_1..x_i..x_n \ \& \ Rx_1..yx_{i+1}..x_n \rightarrow x_i = y)$

Inter-predicate uniqueness constraints are depicted with the circled "u" notation. The simplest case is covered by TUC4.

TUC4 

$\forall x_1 x_2 yz\ (x_1Ry \ \& \ x_1Sz \ \& \ x_2Ry \ \& \ x_2Sz \rightarrow x_1 = x_2)$

For the reader who is familiar with the relational data model, TUC4 may be explained as follows: if a natural join of R and S is performed (by matching values for the object type playing roles R.1 and S.1) then a UC spans the columns linked by the "u". Let the object types be A, B and C (not necessarily distinct), and let the corresponding attributes of the *natural*

join of R and S (denoted R*S) be a, b and c (assume uniquely named). Then the inter-predicate UC shown in Figure 4.5 amounts to a UC spanning columns b and c of the natural join (indicated by underlining the b, c pair).

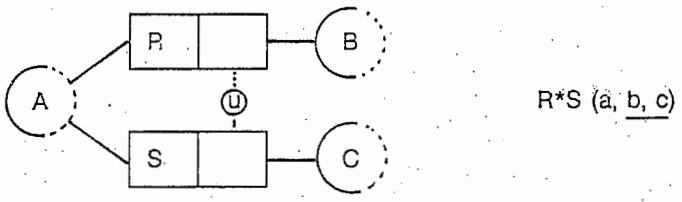
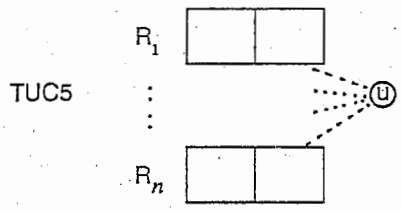


Figure 4.5 An inter-predicate UC in terms of a natural join

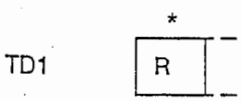
TUC4 is generalized in TUC5 to the case of n binaries.



$$\forall x_1 x_2 y_1 .. y_n \, (x_1 R_1 y_1 \,\&..\&\, x_1 R_n y_n \,\&\, x_2 R_1 y_1 \,\&..\&\, x_2 R_n y_n \rightarrow x_1 = x_2)$$

The possibility of extending the use of the "u" notation to handle further cases (e.g. higher arities, equijoin over more than one attribute, UC over combinations of tuples) suggests itself; currently NIAM diagrams do not cater for such cases.

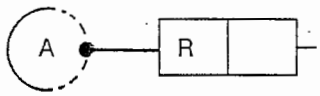
If a *derived predicate* is included on a CS diagram, it is asterisked. Any constraints marked on a predicate that is singly-asterisked may be ignored when translating a CS diagram into KL since, if correct, these constraints are implied by the definition of the derived predicate and the constraints on the other predicates. This is summarized in TD1.



Since constraints may be included on these predicates to make them more obvious to humans, it is still important to ensure that such constraints are consistent with the rest of the CS. These constraints may be translated and checked for consistency in the same way as constraints on other predicates. The next chapter introduces a special class of doubly-asterisked derived predicates: constraints on these should not be ignored.

We now consider *mandatory roles* (total roles). For background discussion on this topic see our earlier work (NH89, sec. 6.3). In this thesis we make some minor changes to our earlier treatment of mandatory roles for derived predicates, and explore the use of strings and numbers in more detail. Roughly, a role is mandatory if, in every interpretation of the CS, it must be played by all the instances of its object type that are mentioned in the interpretation. In contrast to our earlier work, we now allow these instances to be mentioned in stored or derived fact types. A binary case is shown in TMR1.

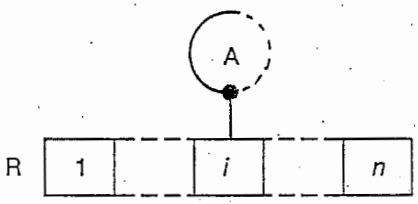
TMR1



$\forall x\ (Ax \rightarrow \exists y\ xRy)$

On a NIAM diagram, a role is explicitly indicated as mandatory by adding a dot where the arc from the role box meets its object type ellipse. TMR2 generalizes TMR1 to predicates of any arity where the role may occur at any position.

TMR2

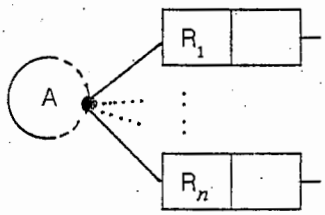


$1 \le i \le n$

$\forall x_i\ (Ax_i \rightarrow \exists x_1..x_{i-1}x_{i+1}..x_n\ Rx_1..x_n)$

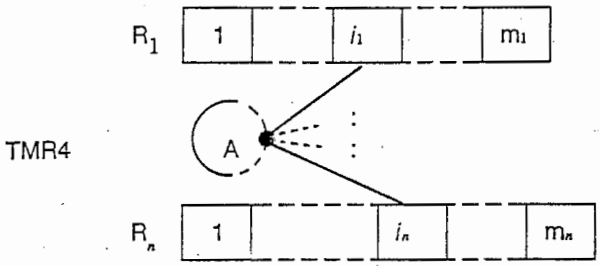
A disjunction of roles is explicitly specified as mandatory by joining the role arcs at the object type ellipse and placing a dot there. TMR3 gives a simple binary case:

TMR3



$\forall x\ [\ Ax \rightarrow \exists y(\ xR_1y \vee .. \vee xR_ny\ )\ ]$

This is generalized in TMR4. Though this general translation is ugly, it is simple to carry out in any individual case.



$$\forall x[\, Ax \rightarrow \exists x_1..x_{m-1}(R_1x_1..x_{i_1-1}xx_{i_1+1}..x_{m_1} \vee .. \vee R_nx_1..x_{i_n-1}xx_{i_n+1}..x_{m_n})]$$

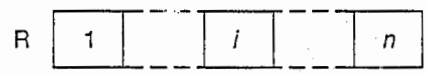
We consider further aspects of mandatory roles (e.g. implicit specification) in the next chapter.

The translation formulae for TMR2 and TMR4 are a little difficult to read (and print!), because in order to talk about a role we had to introduce variables for each position in the predicate. For convenience we now introduce an abbreviated notation which more directly supports the concepts of object types and roles. To begin with, we allow unary predication to be written like a set membership assertion. If x is an IV and A a unary predicate symbol, then we define:

$$x \in A \quad =_{df} \quad Ax$$

Note that this is just a rewrite rule. Although we may informally think of "x $\in$ A" as saying that x is a member of the set A, where in a given interpretation a predicate is identified with its extension, we have not actually included any set theory in our first order formalization. In particular, we are not going to set up an axiomatic basis for sets. Such abbreviations are typically used only for predicates corresponding to NIAM object types, as suggested by the diagram on the right.

We now introduce a more general abbreviation. If R is a n-ary predicate, then we identify its ith role as $R.i$ (cf. qualified field names in Pascal, SQL etc.) and define:



$1 \le i \le n$

$$x \in R.i \quad =_{df} \quad \exists x_1..x_n(Rx_1..x_n \,\&\, x = x_i)$$

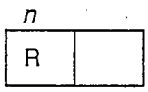
Although informally the diagram helps us picture this in terms of membership in the set of objects in the ith place of the predicate, formally we have just defined an abbreviation. When formal proofs are set out, formulae using the new notation must first be unabbreviated, unless derived inference rules using

the high level notation have been established. However, many formulae may now be written more concisely. For example, the translation rules TMR2 and TMR4 may be replaced with the shorter forms TMR2' and TMR4' as shown. The equivalence between the short and long forms is easily proved by unabbreviating and then using a deduction tree.

TMR2' $\forall x(x \in A \rightarrow x \in R.i)$

TMR4' $\forall x(x \in A \rightarrow x \in R_1.i_1 \vee .. \vee x \in R_n.i_n)$

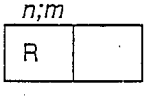
Uniqueness constraints are a special case of occurrence *frequency constraints* (FCs). Recall our discussion of these constraints in section 2.2. An FC of n (where n is an integer) on a role or role sequence means any given instantiation of the role (sequence) occurs n times (in that relation). In contrast to our earlier work, we use $n;m$ to mean an integer in the range $n..m$. An FC of $n;m$ on a role (sequence) means that each instantiation of the role (sequence) occurs at least n and at most m times. A simple binary case is set out in TFC1. This says that if x R's something, x R's exactly (at least and at most) n things.

TFC1 

$n \geq 1$

$\forall x[\exists y_1 xRy_1 \rightarrow \exists y_2..y_n(y_1 \neq y_2 \& y_1 \neq y_3 \&..\& y_{n-1} \neq y_n \& xRy_2 \&..\& xRy_n)]$
$\& \forall xy_1..y_{n+1} [xRy_1 \&..\& xRy_{n+1} \rightarrow y_1=y_2 \vee y_1=y_3 \vee..\vee y_n=y_{n+1}]$

Notice that when $n = 1$, TFC1 reduces to TUC2. A simple binary case for the frequency range constraint is given in TFC2. This says that if x R's something it R's at least n things and at most m things.

TFC2 

$1 \leq n \& n \leq m$

$\forall x[\exists y_1 xRy_1 \rightarrow \exists y_2..y_n(y_1 \neq y_2 \& y_1 \neq y_3 \&..\& y_{n-1} \neq y_n \& xRy_2 \&..\& xRy_n)]$
$\& \forall xy_1..y_{m+1} [xRy_1 \&..\& xRy_{m+1} \rightarrow y_1=y_2 \vee y_1=y_3 \vee..\vee y_m=y_{m+1}]$

By allowing the possibility that $n = m$, an FC of n may be defined as an FC of $n;n$. So TFC1 becomes a special case of TFC2. Clearly, TFC2 may be generalized further to the case where the predicate may be of any arity and the role may be in any position. Though this generalization is

straightforward, its specification is lengthy and difficult to read using our current notation. Indeed, even TFC1 and TFC2 are somewhat difficult to interpret at a glance. To simplify the formulation of such results, we introduce some higher level constructs as abbreviations.

If x and y are IVs and Φx is a wff with no free occurrence of y, then we define $\exists!$ (the quantifier for unique existence) thus:

$$\exists!x\ \Phi x \quad =_{df} \quad \exists x[\ \Phi x\ \&\ \forall y(\Phi y \rightarrow y=x)\ ]$$

The definiens is read "there is exactly one x such that Φx". The "no free occurrence" condition is required to avoid variable collision, e.g. "z" is used instead of "y" in the following unabbreviation:

$$\forall x\exists!y\ xRy \quad =_{df} \quad \forall x\exists y[\ xRy\ \&\ \forall z(xRz \rightarrow z=y)\ ]$$

Similarly, we define a denumerable list of numeric existential quantifiers $\exists^1$, $\exists^2$, $\exists^3$ etc. The individual cases are obtained by replacing n in the following definition schema with 1, 2, etc. Clearly $\exists!$ is just a variant notation for $\exists^1$. Again, Φ must have no free occurrences of y.

$$\exists^n x\ \Phi x \quad =_{df} \quad \exists x_1..x_n[\ \Phi x_1\ \&..\&\ \Phi x_n\ \&\ x_1\neq x_2\ \&\ x_1\neq x_3\ \&..\&\ x_{n-1}\neq x_n\ \&\ \forall y(\Phi y \rightarrow y=x_1 \vee..\vee y=x_n)\ ]$$

For any given n, this is read "there are exactly n x's such that Φx". We may now set out the translation rule TFC1 more concisely as TFC1':

TFC1' $\quad \forall xy(\ xRy \rightarrow \exists^n z\ xRz\ )$

It is also convenient to define a denumerable number of numeric range existential quantifiers $\exists^{0;1}$, $\exists^{0;2}$, .., $\exists^{1;1}$ etc. Individual cases are obtained by replacing n and m with 0, 1, 2 etc. where $n \leq m$.

$$\exists^{n;m} x\ \Phi x \quad =_{df} \quad \exists x_1..x_n[\ \Phi x_1\ \&..\&\ \Phi x_n\ \&\ x_1\neq x_2\ \&\ x_1\neq x_3\ \&..\&\ x_{n-1}\neq x_n\ ]\ \&\ \forall x_1..x_{m+1}[\ \Phi x_1\ \&..\&\ \Phi x_{m+1} \rightarrow x_1=x_2 \vee x_1=x_3 \vee..\vee x_m=x_{m+1}\ ]$$

For a given n and m this is read "there are at least n and at most m x's such that Φx". If $n = 0$ the first conjunct disappears, so this is read as "there are at most m x's such that Φx". For example, "$\exists^{5;7}$x Senator x" is read

"there are at least 5 and at most 7 senators", and "$\exists^{0;7}$x Senator x" is read "there are at most 7 senators". Clearly the earlier numeric existential quantifiers are special cases of the numeric range existential quantifiers (viz. where $n = m$). TFC2 may now be shortened to TFC2':

TFC2' $\forall xy(xRy \rightarrow \exists^{n;m}z\ xRz)$

Uniform lists of $\exists!$ may be abbreviated as for $\forall$ and $\exists$, e.g. "$\exists!xy$ xRy" is short for "$\exists!x\exists!y$ xRy". However, we do not so abbreviate lists of the other quantifiers, e.g. we do not use "$\exists^2xy$ xRy" as short for "$\exists^2x\exists^2y$ xRy". Partly to cater for such cases, we allow a sequence of IVs (e.g. "$x_1x_2x_3$") to be abbreviated as an underlined symbol (e.g. "$\underline{x}$"). Although informally such a symbol may be thought of as denoting a tuple variable, formally we treat it simply as an abbreviation.

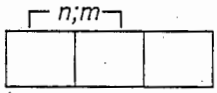
If $\Phi\underline{x}$ is a formula in which $\underline{x}$ only occurs appended to n-ary predicate symbols, and $x_1..x_n$ is a sequence of IVs which are not free in $\Phi\underline{x}$, then the following abbreviations are defined:

$$\begin{aligned}
\forall\underline{x}\ \Phi\underline{x} &=_{df} \forall x_1..x_n\ \Phi x_1..x_n \\
\exists\underline{x}\ \Phi\underline{x} &=_{df} \exists x_1..x_n\ \Phi x_1..x_n \\
\exists!\underline{x}\ \Phi\underline{x} &=_{df} \exists! x_1..x_n\ \Phi x_1..x_n \\
\underline{x} = \underline{y} &=_{df} x_1=y_1\ \&\ ..\ \&\ x_n=y_n
\end{aligned}$$

For example, if R and S are ternary then "$\exists\underline{x}(R\underline{x} \rightarrow S\underline{x}\ \&\ Tx_2)$" may be unabbreviated as "$\exists xyz(Rxyz \rightarrow Sxyz\ \&\ Ty)$". The translation rule TUC3 may now be shortened to the primed form:

TUC3' $\forall x\ [\, x \in R.i \rightarrow \exists!\underline{y}(R\underline{y}\ \&\ y_i=x)\,]$

TFC3 shows a simple case of an FC on a role combination: each instantiation of the role pair marked out by the "$\ulcorner\urcorner$" occurs at least n and at most m times. We first set out the long version of the translation rule.

TFC3 R  $1 \le n\ \&\ n \le m$

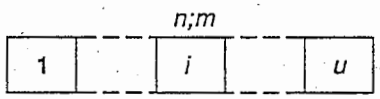
$\forall xy[\exists z_1\ Rxyz_1 \rightarrow \exists z_2..z_n(\, z_1 \neq z_2\ \&\ z_1 \neq z_3\ \&..\&\ z_{n-1} \neq z_n\ \&\ Rxyz_2\ \&..\&\ Rxyz_n\,)]$
$\&\ \forall xyz_1..z_{m+1}[Rxyz_1\ \&..\&\ Rxyz_{m+1} \rightarrow z_1=z_2\ \vee\ z_1=z_3\ \vee..\vee\ z_m=z_{m+1}]$

To simplify this and other cases, we define the usage of the other existential quantifiers with underlined variables in an analogous manner to our earlier definition. All such cases are covered by the following definition (list). Note that $\underline{x}_1$ etc. are underlined variables.

$$\exists^{n;m}\underline{x}\ \Phi\underline{x} =_{df} \exists\underline{x}_1..\underline{x}_n[\ \Phi\underline{x}_1 \&..\&\ \Phi\underline{x}_n \&\ \underline{x}_1\neq\underline{x}_2 \&\ \underline{x}_1\neq\underline{x}_3 \&..\&\ \underline{x}_{n-1}\neq\underline{x}_n\ ]$$
$$\&\ \forall\underline{x}_1..\underline{x}_{m+1}[\ \Phi\underline{x}_1 \&..\&\ \Phi\underline{x}_{m+1} \rightarrow \underline{x}_1=\underline{x}_2 \vee \underline{x}_1=\underline{x}_3 \vee..\vee \underline{x}_m=\underline{x}_{m+1}\ ]$$

The general case of an FC on a role anywhere in any predicate, may now be specified as in TFC4.

TFC4 R



$1 \le n \le m$
$1 \le i \le u$

$$\forall x[\ x \in R.i \rightarrow \exists^{n;m}\underline{z}(\ R\underline{z} \&\ z_i = x\ )]$$

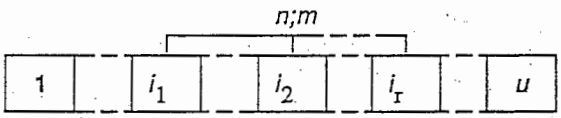
Equivalently: $\forall\underline{x}[\ R\underline{x} \rightarrow \exists^{n;m}\underline{z}(\ R\underline{z} \&\ z_i = x_i\ )]$

The translation rule TFC3 may be shortened to TFC3':

TFC3' $\forall xyz[\ Rxyz \rightarrow \exists^{n;m}\underline{x}(\ R\underline{x} \&\ x_1=x \&\ x_2=y\ )]$

The most general case of an FC may be translated using rule TFC5. Here the predicate is of arity u, and the constraint spans the r roles R.i1,..,R.ir.

TFC5 R



$$\forall\underline{x}\ [\ R\underline{x} \rightarrow \exists^{n;m}\underline{z}(\ R\underline{z} \&\ z_{i_1}=x_{i_1} \&\ ..\ \&\ z_{i_r}=x_{i_r}\ )]$$

It is convenient to use "$\exists?$" as an alternative notation for "$\exists^{0;1}$". If x and y are IVs and Φx is a wff with no free occurrence of y, then we define $\exists?$ (read "there is at most one") thus:

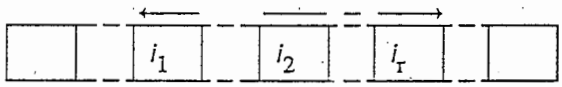
$$\exists?x\ \Phi x =_{df} \forall xy(\ \Phi x \&\ \Phi y \rightarrow x=y\ )$$

This enables a UC on a role to be expressed more concisely, e.g. TUC2' abbreviates TUC2:

TUC2' $\forall x \exists ?y\ xRy$

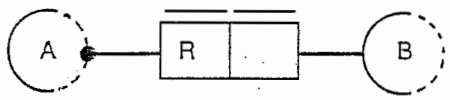
This quantifier is also useful for translating *uniqueness constraints on compound fact types*. Although in a conceptual schema we aim to eliminate compound fact types in favour of elementary fact types, it is useful to be able to discuss compound fact types at the conceptual level. With a compound fact type of arity n, a UC may span between 1 and n roles. The general case (TUC6) is now set out. Here a UC spans r roles in positions i_1, .., i_r. This case actually covers all possible intra-predicate UCs, whether the fact type is elementary or compound.

TUC6 R



$$\forall x_1..x_r \exists ?\underline{z}\ (R\underline{z} \ \&\ z_{i_1} = x_1 \ \&\ ..\ \&\ z_{i_r} = x_r)$$

The combination of a mandatory role constraint and UC on a role may also be rendered concisely by use of $\exists !$. For example, the CS fragment in Figure 4.6 may be translated as the three KL sentences shown.



$\forall xy(xRy \rightarrow Ax \ \&\ By)$
$\forall x(Ax \rightarrow \exists !y\ xRy)$
$\forall y \exists ?x\ xRy$

Figure 4.6 A concise translation of a CS fragment

4.3 Subtypes

We now consider how *subtype* constraints may be mapped into KL. For a detailed but high level treatment of NIAM subtypes see our earlier work (NH89, Ch. 6). In CS diagrams, subtypes are connected to their immediate supertype by a directed line segment. The information conveyed by this linkage may be translated as shown in Figure 4.7:

$\forall x(Bx \rightarrow Ax)$

Figure 4.7 Information conveyed by a subtype link

However, in NIAM only well defined proper subtypes are allowed. This means that included in the CS there must be *subtype definition* (SD). Such definitions are written in textual form below the CS diagram. Since the subtype definitions imply the linkage information there is no need to translate any subtype linkages into KL. However, the subtype definitions must be translated. In order for such translation to be automatic, a formal language for such definitions needs to be defined, together with an algorithm for mapping to KL. As a temporary solution, for this thesis we assume that, except for string and numeric subtypes (as explained shortly), the definitions are actually given in KL itself; so such definitions are mapped unaltered. For example, a subtype Man might be defined as follows:

$$\forall x [\text{Man } x \equiv \exists y(x \text{ has\_gender } y \ \& \ y \text{ has\_gendercode 'm'})]$$

Membership in the relevant supertype (e.g. Person x) is implied by the typing constraint on the predicate "... has_gender ...", but could be added as a comment in braces for human readers. A useful exercise would be to develop a more convenient notation for such definitions. For example, given that Gender is standardly referenced by gendercode it would not be difficult to define translation rules so that the following text would be treated as an abbreviation of the previous definition: Man $=_{df}$ Person having Gender 'm'. It should be understood that the main impact of adding subtype definitions is felt only when the typing constraints are defined for the roles played by the subtype (consider rule TPN where the subtype is one of the A_i).

Subtypes for strings and numbers are usually specified in a different way. In our earlier work (NH89, secs 6.2, 7.1) we introduced some notations for this task. We now provide a formal and improved treatment of these notations, and clarify the meaning of abbreviated 1:1 reference schemes as portrayed on NIAM diagrams. We start by examining *lexical subtypes*, i.e. subtypes of String.

Recall that lexical types appear as broken ellipses on a NIAM diagram. Sometimes it may be convenient to include names for these types on the

diagram, but this is not essential. If no lexical constraint is specified for the type then it is assumed to be the type String (see rule TBE). If it is desired to restrict the type to a proper subset of String, then an appropriate subtype definition is written, in abbreviated form, beside the subtype node (or even inside it if the node is unnamed). We now provide a standard list of abbreviations for specifying such lexical subtypes. These cover almost all practical cases. Any other cases may be defined using the String axioms established earlier.

Earlier we defined expressions of the form x ϵ A to mean Ax. It is convenient to define several other uses of ϵ. If x is an individual variable and $a_1,..,a_n$ are either n string constants or n numeric constants then we define:

$$x \epsilon \{a_1,..,a_n\} \quad =_{df} \quad x = a_1 \vee .. \vee x = a_n$$

If s_1, .., s_n are string constants, a lexical object type may be specified as the set $\{s_1,..,s_n\}$ by listing this set beside the node, i.e.

TSD1  $\{s_1,..,s_n\}$ $\forall x[\, Ax \equiv x \epsilon \{ s_1,..,s_n \}\,]$

For example, Gendercode might be specified as {'m','f'} and Colourcode as {'red','green','blue'}.

The lexical subtypes Digit, Letter, Digits and Letters have already been defined (Axioms ST13..16). We now use these to specify various *string patterns*. Angle brackets are used to delimit such lexical subtype definitions. We first define uses of ϵ with angle brackets. Here n is a positive integer.

$x \epsilon \langle cn \rangle \quad =_{df} \quad$ String x & x≠'' & len(x) ≤ n
$x \epsilon \langle nc \rangle \quad =_{df} \quad$ String x & len(x) = n
$x \epsilon \langle dn \rangle \quad =_{df} \quad$ Digits x & len(x) ≤ n
$x \epsilon \langle nd \rangle \quad =_{df} \quad$ Digits x & len(x) = n
$x \epsilon \langle an \rangle \quad =_{df} \quad$ Letters x & len(x) ≤ n
$x \epsilon \langle na \rangle \quad =_{df} \quad$ Letters x & len(x) = n

These respectively mean that x is a non-null string of at most n characters, exactly n characters, at most n digits, exactly n digits, at most n letters,

and exactly *n* letters. For the exact cases we allow the "c", "d" or "a" to be written *n* times as a "*vivid* pattern" e.g.

<cc> may be used instead of <2c>
<dddd>may be used instead of <4d>
<aaa> may be used instead of <3a>
etc.

Vivid string patterns may be (recursively) concatenated by juxtaposition, i.e. if <*p*> and <*q*> are vivid string patterns then so is <*pq*>, and

$$x \in \langle pq \rangle =_{df} \exists yz(x = y+z \ \& \ y \in \langle p \rangle \ \& \ z \in \langle q \rangle)$$

Only vivid string patterns may be concatenated by juxtaposition. However, all string patterns may be (recursively) concatenated by use of +, i.e. if <*p*> and <*q*> are string patterns then so is <*p*+*q*>, and

$$x \in \langle p+q \rangle =_{df} \exists yz(x = y+z \ \& \ y \in \langle p \rangle \ \& \ z \in \langle q \rangle)$$

If <..> is a string pattern, then a lexical node may be specified as this string subtype by writing the pattern beside the node, i.e.

TSD2 (A) <..> $\forall x[Ax \equiv x \in \langle .. \rangle]$

For example, the left subtype in Figure 4.8 might be used for subject codes (e.g. "CS112"). Instead of this vivid form, the string pattern could have been specified as <2a+3d>. The right subtype in this figure might be used for names consisting of a surname of one to 20 characters, then a space, then one to three initials.

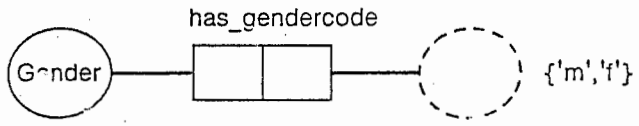
(Subjectcode) <aaddd> (Personname) <a20+' '+a3>

$\forall x$ [Subjectcode x ≡ String x & $\exists yz$(x = y+z & Letters y & len(y)=2 & Digits z & len(z)=3)]

$\forall x$ [Personname x ≡ String x & $\exists yz$(x = y+' '+z & Letters y & len(y) ≤ 20 & Letters z & len(z) ≤ 3)]

Figure 4.8 Two examples of lexical constraints

In setting out subtype specification rules, we use a predicate variable to refer to the subtype. *If no subtype name is supplied, the subtype definition is used instead of this name when typing constraints are specified for the roles played by the subtype.* For example, consider Figure 4.9.



$\forall x,y$ (x has_gendercode y → Gender x & y = 'm' ∨ y = 'f')

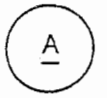
Figure 4.9 Example of an unnamed lexical subtype

If we had included the name "Gendercode" for the string subtype, then the subschema would have been translated by TPB and TSD1 to the two formulae: ∀xy(x has_gendercode y → Gender x & Gendercode y); ∀x(Gendercode x ≡ x='m' ∨ x='f').

Note that on a CS diagram, lexical subtypes often overlap (e.g. <c20>, <aaddd>, {'Spring', 'Summer', 'Autumn', 'Winter'}). This overlapping is implicitly specified by lexical constraints of the kind just discussed. It is very rare to exhibit any arrows between lexical types. If such arrows are shown they may be ignored in translating since they are implied by the subtype definitions.

The same comment applies to *numeric subtypes*. Earlier, we specified four subtypes of Number: Real, Integer, Cardinal and Posint. If used, these appear as a solid ellipse with the appropriate name: subtype linkages to Real are not normally shown on the diagram. If any other subtype of Real is to be used it is shown as a (possibly named) solid ellipse with the subtype definition written, in abbreviated form, beside or in the ellipse; again, a subtype arrow to Real is usually omitted. We now provide some standard abbreviations for specifying such subtypes. If needed, other real subtypes may be defined using the Real axioms established earlier.

As with strings, a finite set of real numbers may simply be listed in full. The translation rule is TSD3 (cf. TSD1), where n_1, .., n_m are numeric constants for the Reals.

TSD3  $\{n_1,..,n_m\}$ $\forall x(Ax \equiv x \in \{n_1,..,n_m\})$

Recall that quotes must not be used, otherwise the constants denote strings rather than numbers. For example, {1,2,3} is a set of numbers but

{'1','2','3'} is a set of strings. One formation rule for schema diagrams is that sets of quoted constants may be specified only for broken ellipses.

Other subtypes of Real are delimited by square brackets or round brackets, depending on the case. We first define the use of ϵ with such bracketed expressions. In these definitions, n and m are integers where $n < m$, and r and s are reals where $r < s$. Integer subranges are defined with the help of "..", e.g. [1..5] is {1,2,3,4,5}. We use "_" (suggesting a segment of the real number line) to do a similar job for a continuum of reals, e.g. [0_1] is the set of reals from 0 through 1. The dn notation informally means a number denoted by a non-null string of up to n digits, and ± allows signed numbers. A round bracket is used to exclude a delimiting number, e.g. [0_1) is the set of non-negative reals below 1. Since this use of round brackets requires that an underscore be included, there is no danger of conflation with our use of parentheses as pair delimiters.

$x \in [n..m]$	$=_{df}$	Integer x & $x \geq n$ & $x \leq m$
$x \in [n..]$	$=_{df}$	Integer x & $x \geq n$
$x \in [..n]$	$=_{df}$	Integer x & $x \leq n$
$x \in [r\_s]$	$=_{df}$	Real x & $x \geq r$ & $x \leq s$
$x \in [r\_]$	$=_{df}$	Real x & $x \geq r$
$x \in [\_r]$	$=_{df}$	Real x & $x \leq r$
$x \in (r\_s)$	$=_{df}$	Real x & $x > r$ & $x < s$

Other cases for round brackets, and permutations of round and square brackets, may obviously be specified.

If a maximum precision needs to be specified (e.g. dollar values), this may be catered for as follows. However, any associated derivation rules need to have corresponding restrictions. For each of these cases we permit "d..d", where there are n "d"s, as a vivid variant of "dn".

$x \in [.dn]$	$=_{df}$	Real x & $x \geq 0$ & $x < 1$ & Integer $x*10^n$
$x \in [dn.dm]$	$=_{df}$	Real x & $x \geq 0$ $x < 10^n$ & Integer $x*10^m$
$x \in [\pm.dn]$	$=_{df}$	Real x & $x > -1$ & $x < 1$ & Integer $x*10^n$
$x \in [\pm dn.dm]$	$=_{df}$	Real x & $x > -10^n$ & $x < 10^n$ & Integer $x*10^m$

For example, real numbers in the range -99999.99 through 99999.99 with at most 2 digits after the decimal point may be specified as [±d5.d2] or [±ddddd.dd]. We regard it to be an external than a conceptual issue whether leading or trailing zeros are needed for input or output (e.g. in our treatment 3, 3.0, 00003.00 etc. are all defined as 1+1+1).

A union of ranges may be (recursively) specified by separating the ranges by commas, i.e. if $[r_1]$ and $[r_2]$ are real ranges, then so is $[r_1,r_2]$ and

$$x \in [r_1,r_2] \quad =_{df} \quad x \in [r_1] \vee x \in [r_2]$$

Here we have used "[", "]" generically to include round brackets. If [..] is any numeric constraint of the kind specified with our bracket notation, then a node may be specified as the corresponding subtype of Real by writing this notation inside or beside the node, i.e.

TSD4 

[..] $\forall x(Ax \equiv x \in [..])$

Figure 4.10 presents two examples, where the ϵ notation has been unabbreviated. As indicated earlier, it is optional whether subtypes of Real are given names.

[1..7] $\forall x(Rating\_nr\ x \equiv Integer\ x \,\&\, x \geq 1 \,\&\, x \leq 7)$

(0_] $\forall x(PosReal\ x \equiv Real\ x \,\&\, x > 0)$

Figure 4.10 Two examples of numeric subtype definitions

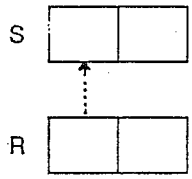
We end this section by defining a further abbreviation which is useful later. If x is a term and S is a predicate name, role name or a bracketed range specification, then we define the symbol "$\notin$" (read informally as "does not belong to") by the following abbreviation:

$$x \notin S \quad =_{df} \quad \sim(x \in S)$$

4.4 Subset, equality and exclusion constraints

Background on subset, equality and exclusion constraints may be found in NH89 (sec. 8.2). We now express these in KL, starting with *subset constraints*. These are marked with a dotted arrow pointing to the superset. A simple case is TSC1: both raw and role versions are given.

TSC1

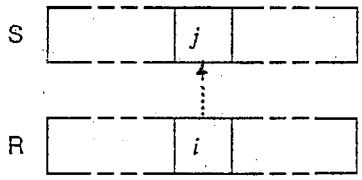


$\forall x (\exists y\, xRy \rightarrow \exists z\, xSz)$

$\forall x (x \in R.1 \rightarrow x \in S.1)$

This is generalized in TSC2, where the predicates may be of any arity (including unary) and the roles may be in any position. Despite visual appearances, we do not require the predicates to be of the same arity or the roles to be in the same position; moreover, R and S need not be distinct.

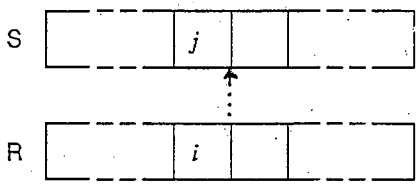
TSC2



$\forall x (x \in R.i \rightarrow x \in S.j)$

TSC3 covers the general case of a subset constraint between pairs of contiguous roles. If R and S are binary this simplifies to $\forall xy(xRy \rightarrow xSy)$. Note that predication is needed to establish the underlined variables as "row variables".

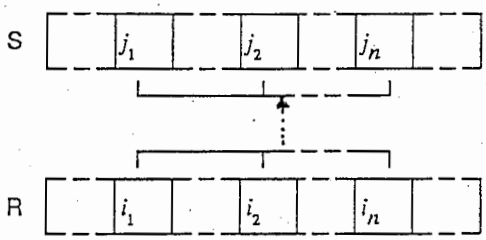
TSC3



$$\forall xy[\ \exists \underline{z}(\ R\underline{z}\ \&\ x = z_i\ \&\ y = z_{i+1}\ ) \rightarrow \exists \underline{w}(\ S\underline{w}\ \&\ x = w_j\ \&\ y = w_{j+1}\ )]$$

This generalizes further to TSC4, the case of a subset constraint between predicate tuples of (not necessarily contiguous) roles:

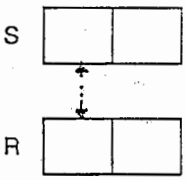
TSC4



$$\forall x_1..x_n[\exists \underline{y}(R\underline{y} \,\&\, x_1 = y_{i_1} \,\&..\&\, x_n = y_{i_n}) \rightarrow \exists \underline{z}(S\underline{z} \,\&\, x_1 = z_{j_1} \,\&..\&\, x_n = z_{j_n})]$$

Equality constraints occur when there are subset constraints in both directions. A dotted line with arrow tips at both ends is shown connecting the relevant operands (optionally, both arrows tips may be deleted). By replacing "→" in TSC1-4 with "≡" we obtain TEC1-4:

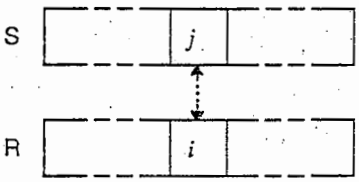
TEC1



$\forall x (\exists y \; xRy \equiv \exists z \; xSz)$

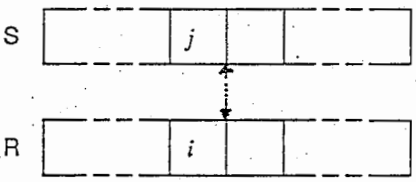
$\forall x (x \in R.1 \equiv x \in S.1)$

TEC2



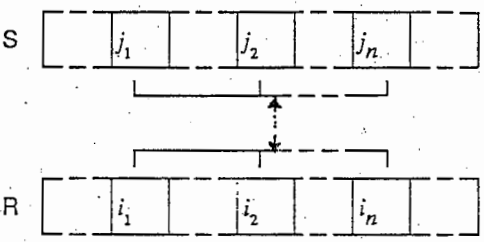
$\forall x (x \in R.i \equiv x \in S.j)$

TEC3



$$\forall xy[\exists \underline{z}(R\underline{z} \,\&\, x = z_i \,\&\, y = z_{i+1}) \equiv \exists \underline{w}(S\underline{w} \,\&\, x = w_j \,\&\, y = w_{j+1})]$$

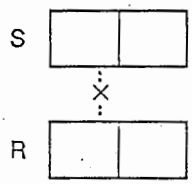
TEC4



$$\forall x_1..x_n[\exists \underline{y}(R\underline{y} \,\&\, x_1 = y_{i_1} \,\&..\&\, x_n = y_{i_n}) \equiv \exists \underline{z}(S\underline{z} \,\&\, x_1 = z_{j_1} \,\&..\&\, x_n = z_{j_n})]$$

We now consider *exclusion constraints*. A dotted line connects the relevant operands to an exclusion mark "×" (optionally, this mark may be circled). The simple case of TXC1 is given in both raw and role versions. The role versions are used for the other cases TXC2-4.

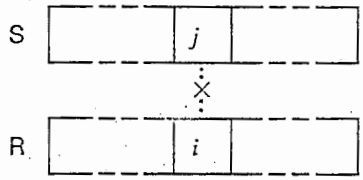
TXC1



$\forall xyz\sim(xRy \;\&\; xRz)$

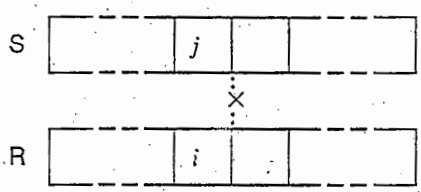
$\forall x\sim(x \in R.1 \;\&\; x \in S.1)$

TXC2



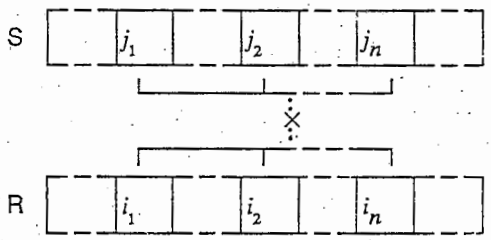
$\forall x\sim(x \in R.i \;\&\; x \in S.j)$

TXC3



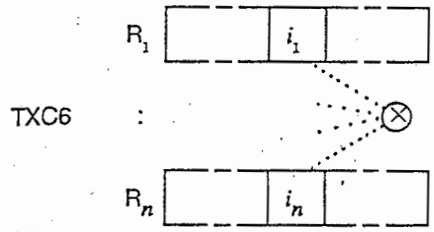
$\forall xy\sim[\exists \underline{z}(R\underline{z} \;\&\; x = z_i \;\&\; y = z_{i+1}) \;\&\; \exists \underline{w}(S\underline{w} \;\&\; x = w_j \;\&\; y = w_{j+1})]$

TXC4



$\forall x_1..x_n\sim[\exists \underline{y}(R\underline{y} \;\&\; x_1 = y_{i_1} \;\&\; .. \;\&\; x_n = y_{i_n}) \;\&\; \exists \underline{z}(S\underline{z} \;\&\; x_1 = z_{j_1} \;\&..\&\; y_n = z_{j_n})]$

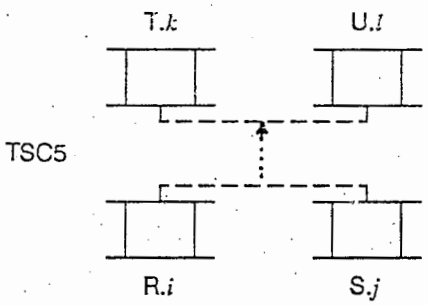
TXC1-4 cover the cases where there are two operands. These generalize respectively to TXC5-8, where there is a mutual exclusion constraint among n operands. When n exceeds 2, it is generally recommended that the "×" mark is circled. The most important case in practice is where we have a set of mutually exclusive roles (see TXC6: clearly this subsumes TXC5). Here no object can simultaneously instantiate any role pair selected from the n roles, i.e a simple exclusion constraint exists between each of the role pairs.



$\forall x \sim(x \in R_1.i_1 \,\&\, x \in R_2.i_2 \vee x \in R_1.i_1 \,\&\, x \in R_3.i_3 \vee .. \vee x \in R_{n-1}.i_{n-1} \,\&\, x \in R_n.i_n)$

Similarly, the formalizations of TXC1,3-4 may be adapted to provide formalizations of TXC5,7-8. Since these extensions are straightforward, we do not specify them here.

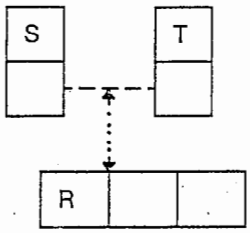
We conclude this section by introducing *join subset*, *join equality* and *join exclusion constraints*, as an extension to NIAM. The pairwise subset case is set out in TSC5. As the general specification in KL is awkward, we convey the meaning of the constraint using relational concepts. Here we have four (not necessarily distinct) predicates R, S, T and U. $R*S[c_i,c_j]$ is the *projection* on columns c_i, c_j of the *natural join* of R and S. Column c_i is the column that role R.*i* maps onto in the join, and so on. Basically the constraint asserts that the set of pairs formed in the join projection $R*S[c_i,c_j]$ is a subset of the set of pairs formed in the join projection $T*U[c_k,c_l]$. Clearly, for non-vacuous applications the pairs being compared must belong to compatible pair types (e.g. roles R.*i* and T.*k* might be played by object type A, and roles S.*j* and U.*l* played by object type B).



$R*S[c_i,c_j] \subseteq T*U[c_k,c_l]$

Similarly, translations may be set out for a join equality constraint (TEC5) and a join exclusion constraint (TXC9). Clearly, the TSC5, TEC5 and TXC9 pairwise join constraints may be generalized to the tuple-wise case. Although the specification of such general translation rules is awkward in KL, for any specific case the KL definition is straightforward. For example, TEC5a is a simple case of TEC5: this is later used in discussing transformations on compositely described object types; here the population of the (S.2,T.2) join pairs must equal the population of the (R.1,R.2) pairs.

TEC5a

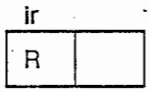


$\forall xy[\ \exists z(zSx \ \&\ zTy) \equiv \exists w\ Rxyw]$

4.5 Homogeneous binaries and other constraints

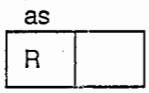
Background on homogeneous binaries and other constraints may be found in NH89 (secs 8.3, 8.4). Irreflexive, asymmetric and intransitive constraints are translated for simple homogeneous binaries as shown:

TIR1



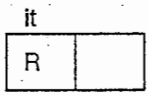
$\forall x \sim xRx$

TAS1



$\forall xy(\ xRy \rightarrow \sim yRx\ )$

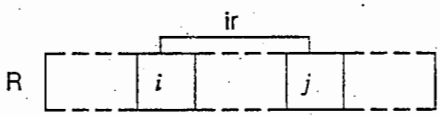
TIT1



$\forall xyz(\ xRy\ \&\ yRz \rightarrow \sim xRz\ )$

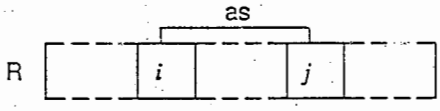
In some cases, similar constraints need to be applied to embedded homogeneous role pairs (e.g. Part contains Part in Quantity). These are catered for by the following rules. Here R is at least a ternary.

TIR2

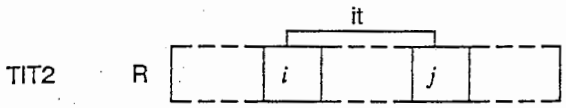


$\forall \underline{xy}\ [\ R\underline{y} \rightarrow \sim(\ x = y_i\ \&\ x = y_j\ )]$

TAS2

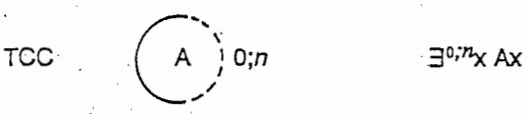


$\forall \underline{xyzw}\ [\ R\underline{z}\ \&\ R\underline{w}\ \&\ x = z_i\ \&\ y = z_j \rightarrow \sim(\ y = w_i\ \&\ x = w_j\ )]$



$\forall xyzwvu\ [\ R\underline{w}\ \&\ R\underline{v}\ \&\ R\underline{u}\ \&\ x=w_i\ \&\ y=w_j\ \&\ y=v_i\ \&\ z=v_j \rightarrow \sim(x=u_i\ \&\ z=u_j)]$

In some methodologies (e.g. Entity-Relationship modelling) the term "cardinality" is used in classifying relationship types as 1:1, 1:*n*, *n*:1, or *n*:*m*, but NIAM describes such cases in terms of uniqueness constraints. In some NIAM dialects, the term "cardinality constraint" is a synonym for "frequency constraint". However, in our version of NIAM, a *cardinality constraint* may be imposed on an object type to limit the number of members for *each population* of that type (in some cases this might be less than the cardinality of the type, e.g. a type may include many objects only some of which may be used at a time). Such constraints may be indicated by writing the cardinality range beside the object type ellipse, using our semicolon notation, and are easily translated using our numeric existential quantifiers. To specify that there are at most *n* objects in each population of *A*:



For example, to assert that there is only one President, we may write "0;1" beside the object type ellipse for President.

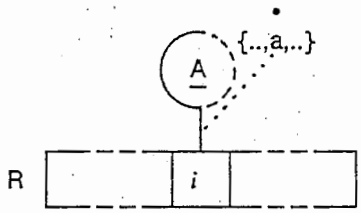
To simplify discussion of the next constraint category, as well as later transformation theorems, we introduce a further metasymbol. When an object type must be either *lexical or numeric* it is shown as a *half-solid circle* with the *type name underlined*:

In some cases we need to specify a constraint on the playing of a role by a single object: we call this a *role-object constraint*. We now define three kinds of role-object constraints for the simple case where the the role is played by an enumerated lexical or numeric object type (similar constraints for described object types are defined in the next chapter). These are set out in TROC1-3. Here *a* is either a string or numeric constant. In each case

the object *a* is connected by a broken arc to the relevant role arc: if it is known that the object type A plays only this role in the global schema diagram then the broken arc may be omitted.

The dot above *a* in TROC1 specifies that if any object plays the indicated role then *a* does. In this sense, *a* might be called a "mandatory object"; we introduced this kind of constraint in NH89 (p. 195) using the term "mandatory entity".

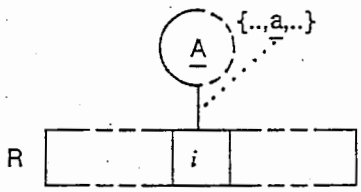
TROC1



$\exists x(x \in R.i) \rightarrow a \in R.i$

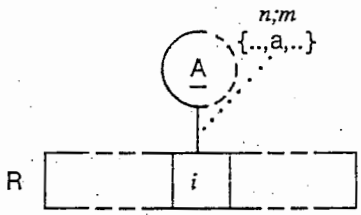
In TROC2, underlining *a* specifies a "restricted uniqueness constraint" for *a* in that role. A frequency range annotation above or below *a* specifies a "restricted frequency constraint" for *a* in that role (TROC3). These constraint categories are introduced here for the first time.

TROC2



$\exists ?\underline{x}(R\underline{x} \,\&\, x_i = a)$

TROC3



$\exists^{n;m}\underline{x}(R\underline{x} \,\&\, x_i = a)$

As a simple example, consider the subschema of Figure 4.11. This might be used to record whether a committee member is the President, one of two possible Vice-Presidents, the Secretary, the Treasurer, or an Ordinary member. The dot over "P" might be used to specify that the President is elected first. The underscores for "P", "S" and "T" indicate there is at most one President, at most one Secretary, and at most one Treasurer. The frequency notation above "VP" indicates there can be at most 2 Vice-Presidents. Since only one role is shown connected to the right-hand object type, broken arcs to this role are assumed.

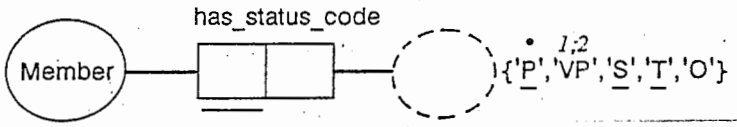


Figure 4.11 A subschema with five role-object constraints

Our constraint specification has now covered all the graphic, static constraint categories of NH89. Since KL has the expressiveness of predicate logic, other kinds of constraint can be asserted in KL, but since this power is open ended we cannot exhaustively list all such cases. Some practical examples of textual constraints (e.g. people cannot die before they are born) are considered in later chapters.

4.6 Nesting

Background on "nested fact types" can be found in NH89. There we adopted the usual high level approach to nesting by defining an "objectified relationship" as a relationship which is also treated as an object which itself plays roles. In this section we formalize nesting in a more general way using our pair function, and offer a simple graphic notation.

To help explain our new approach, we use a familiar example (see Figure 4.12). For simplicity, reference modes have been omitted. The subschema on the left is in the old notation. We have an embedded many:many binary relationship type: Person enrols in Subject. At the outer level we have a functional relationship type for recording the rating scored by a person in a subject. In our analysis (see NH89 p. 100), the embedded part must have a UC spanning its entire length. If it has a shorter key then, if nesting is to occur, this key should be embedded instead. For a contrary view on this matter, see Falkenberg (1986, p. 7-23).

With the old notation, the UC must be marked separately and an ellipse is drawn around the embedded part. This can be awkward to draw, especially if the embedded part has several roles. While we still permit the old notation, we now prefer to use a *frame* ("rounded rectangle") instead of an ellipse, with the understanding that a *full length UC is assumed* (visually one may imagine the UC overlaid on one of the long sides of the frame). Apart from a mandatory role dot, any constraint marks around the embedded part are understood to apply to the inner roles.

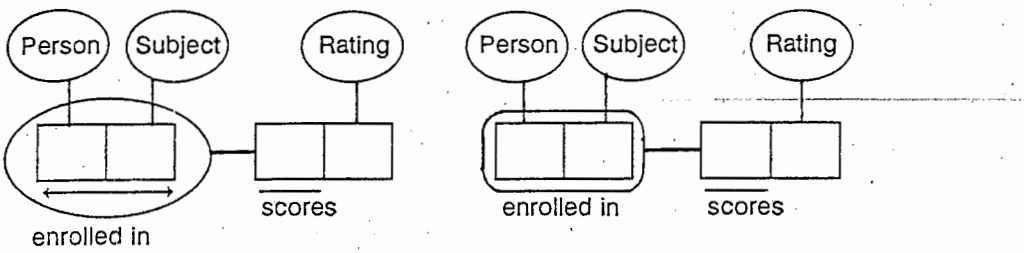


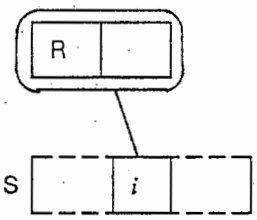
Figure 4.12 Old (left) and new (right) notations for nesting

In NIAM the embedded objects are usually considered to be relationships or facts. However, we now believe it is simpler, and usually just as natural, to think of these objects as pairs. For example, rather than saying that a particular enrolment relationship between a person and a subject scores a rating, we may simply say that the (Person,Subject) pair scores the rating. A major pragmatic reason for adopting this new approach is that it considerably simplifies our treatment of equivalence transformations involving nesting.

Note that we use the word "relationship" in its normal logical sense, i.e. a relationship is a proposition, not a tuple. Relationships may be assigned truth values but individuals cannot. In short, we do not include relationships as individuals. Syntactically, "≡" is an operator between wffs and "=" is an operator between terms, and a wff is not a term. For example, we may say "xRy ≡ xSy" and "(x,y) = (w,z)", but not "xRy = xSy" or "(x,y) ≡ (y,z)".

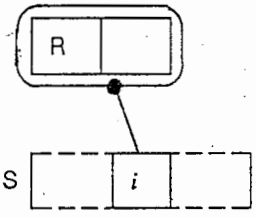
Unless there is a good reason for doing so, we prefer not to name embedded object types. In this case, further translation rules are needed to formalize typing or mandatory role constraints on the role played by the pair type. The binary case is specified by rules TNB1 and TMR5.

TNB1



$\forall x[\, x \in S.i \rightarrow \exists yz(yRz \,\&\, x = (y,z))\,]$

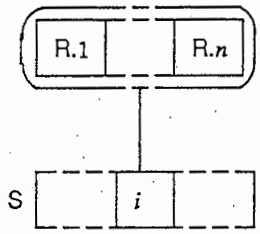
TMR5



{ mandatory role aspect: }
$\forall xy[\, xRy \rightarrow (x,y) \in S.i\,]$

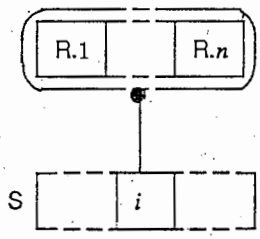
Note that the relevant pairs must instantiate R: it is not enough that they belong to the Cartesian product of the object types attached to R.1 and R.2. The binary case is by far the most common. It generalizes to the case where R is *n*-ary, $n > 1$, as shown in TN2 and TMR6.

TNB2



$$\forall x[x \in S.i \rightarrow \exists x_1..x_n(Rx_1..x_n \;\&\; x = (x_1,..,x_n))]$$

TMR6

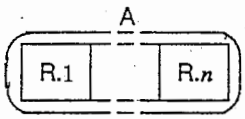


{ mandatory role aspect: }

$$\forall x_1..x_n[\, Rx_1..x_n \rightarrow (x_1,..,x_n) \in S.i\,]$$

Although informally we may think of an object $(x_1,..,x_n)$ as an *n*-ary tuple, formally we capture it as the pair $(x_1,(x_2,..,x_n))$. If the pair object type is named, then the typing and mandatory role constraints are specified in same way as for other named object types. However, the translation of the naming must be specified as follows. We cite only the general case here (TN3).

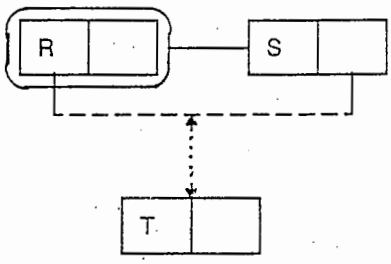
TN3



$$\forall x[\, Ax \equiv \exists x_1..x_n(Rx_1..x_n \;\&\; x = (x_1,..,x_n))\,]$$

Having formalized nesting, we introduce a notation for specifying subset, equality and exclusion constraints for nested fact types where an operand's roles involve both inner and outer predicates. The most important case is TEC6. Note that because of the typing constraint on S, the right-hand side of the quantified equivalence may be replaced by "$\exists z(xRz \;\&\; (x,z)Sy)$". This kind of equality constraint is relevant to licencing certain kinds of schema transformations on nested fact types. Analogous subset (TSC6) and exclusion (TXC10) constraints may also be defined. These pairwise constraint categories may be generalized to tuple-wise cases.

TEC6



$\forall xy[\, xTy \equiv \exists z\, (x,z)Sy\,]$

Before ending this section, we note that nesting can be formalized without the pair function by introducing special functions for each case. For example, one might define the function enrolment(x,y) as a 1:1 mapping between the (Person,Subject) pairs instantiating the Enrolment fact type and its set of values. This function may then be used to refer to the embedded objects. We used this alternative approach in an earlier formalization. However, while closer to the traditional concept of "objectifying a relationship", this alternative complicates later work on equivalence transformations. We feel that our current approach provides a simple and intuitive solution.

We have now formalized most of the basic NIAM graphic notations. In the next chapter we look at more advanced aspects of formalization, and introduce further enhancements to the methodology.

5 Further aspects of NIAM knowledge bases

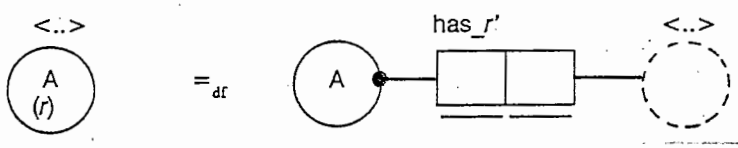
5.1 Reference schemes and numbers

In this chapter we discuss further issues regarding the translation of NIAM knowledge bases into KL, and modify NIAM to provide an improved treatment in certain areas. This section examines reference schemes, focussing on abbreviated notations, and reference schemes that make use of numbers. Background discussion on reference schemes may be found in NH89 (Ch. 7).

In a knowledge base, described entities are never denoted by individual constants. Instead definite descriptions are used (e.g. "the Lecturer with surname 'Halpin'", "the Length with cm_value 175"). Such descriptions are defined in terms of NIAM reference schemes. The final section of this chapter examines definite descriptions in more detail.

Set theoretically, the simplest reference scheme for an entity type is an *injection* (1:1-into mapping) from the population of that type to String. In NH89, we introduced the notion of a *reference mode*, i.e. the manner in which the string relates to the entity being referenced. A meaningful name may be chosen for the reference mode and then written in parentheses beside the name of the entity type. A reference mode name is a sequence of one or more identifier_characters (see Ch. 3), usually starting with a lower-case letter unless this conflicts with standard conventions (e.g. "MHz", "K", "%", "$"). A parenthesized reference mode is just an *abbreviation* for the explicit reference scheme under discussion, as shown in Figure 5.1.

In this figure, *r* is the name of the reference mode, while *r'* is its *expanded form*. If *r* is one of "name", "code" or "title" then *r'* is the result of prepending lc(A), the lowercase version of name of the entity type A, to *r*, else *r'* is simply *r*. For example, "Person (surname)" involves the *reference predicate* "... has_surname ..." while "City (name)" involves the predicate "... has_cityname ...". In the figure, "<..>" denotes any lexical constraint (e.g. <aaddd>, {'m','f'}): writing it beside the entity type indicates it is to be applied to the implicit label type. If it is desired to name the label type, then *r'* is chosen except that its first character is capitalized (e.g., "Surname", "Cityname").



If $r \in$ {'name','code','title'} then $r' = lc(A)+r$ else $r' = r$

Figure 5.1 An abbreviation for a simple reference scheme

One formation rule for CS diagrams is that predicate names be unique: this implies that the (expanded) names of reference modes must also be unique. The expansion scheme in Figure 5.1 conveniently allows multiple occurrences of "name", "code" and "title" on the same diagram, which expand differently, e.g. "Subject (code)" and "Gender (code)" expand to "Subject (subjectcode)" and "Gender (gendercode)". While this enables such abbreviations to be expanded independently of others, in practice shorter expansions may be chosen so long as they are unique to the global schema, e.g. "Gender (code)" might be expanded to "Gender (gcode)" if "gcode" is not used elsewhere.

We define only one abbreviation scheme for predicate names. The word "has" may occur more than once as the abbreviated name of a binary predicate. To preserve uniqueness of predicate names it is always expanded by prepending it to an underscore followed by the lower case version of the name of the adjacent object type (see Figure 5.2).

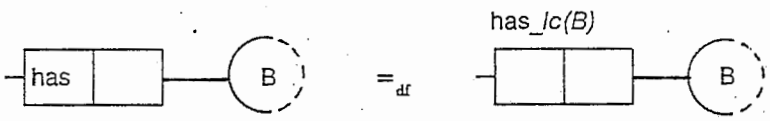


Figure 5.2 "has" may be used as an abbreviated predicate name

For example, the left hand schema fragment shown in Figure 5.3 is an abbreviation of the right hand fragment:

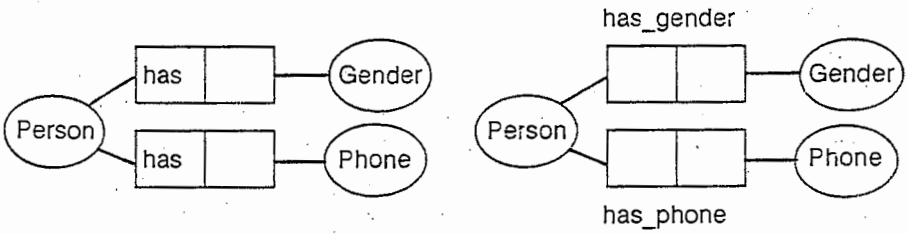


Figure 5.3 The left diagram abbreviates the right diagram

We now consider abbreviations for cases where a combination of two or more labels is used to refer to an entity. For example, a person might be identified using a combination of surname and initials, where the surname

string and the initials string are two separate objects. The general abbreviation schema for a *composite reference scheme* using n labels is given in Figure 5.4. Here $r_1,..,r_n$ are the names of the reference modes, $r_1',..,r_n'$ are their expanded names, and $c_1,..,c_n$ are the respective lexical constraints. In contrast to our earlier work (NH89), we use commas instead of "+"s as separators. The label types are not necessarily distinct, and may be named if desired. If no lexical constraints are given, then each label type is String.

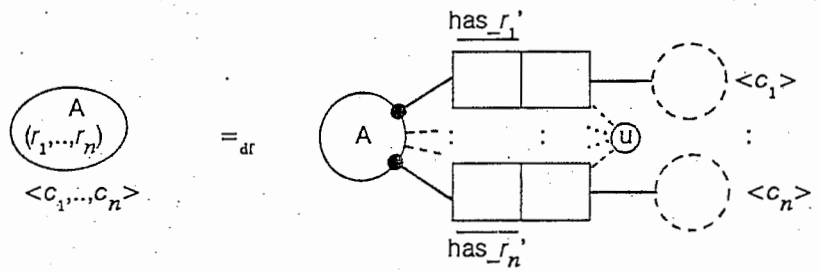


Figure 5.4 Abbreviation of a composite reference scheme

A simple example is given in Figure 5.5.

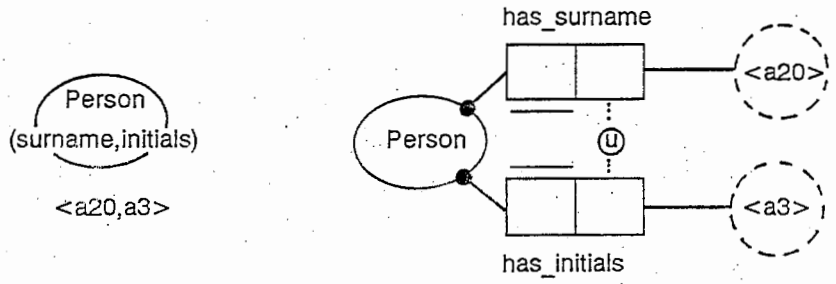


Figure 5.5 The left diagram abbreviates the right diagram

Unlike some other NIAM notations, for any given entity type we never allow more than one reference scheme to be abbreviated. For example, if there is an injection from Subject to Subjectcode and another injection from Subject to Subjecttitle then only one of these may be abbreviated (using the scheme of Figure 5.1). So a commalist of reference modes always means composite reference, rather than a list of simple references.

Although conceptually we make no distinction between the abbreviated and explicit reference forms, in practice we always choose the abbreviated form to indicate the primary reference scheme. We treat selection of a primary reference scheme from candidate reference schemes as an implementation concern rather than as a conceptual concern. As will become more apparent when we discuss derived predicates, this is not the only feature of a NIAM "conceptual" schema diagram which is captured only at a subconceptual level (e.g. a high implementation level).

Injective *numeric reference schemes* may also be abbreviated. In contrast to our earlier work (NH89), the names of numeric reference modes are *underlined*. The reference mode with abbreviated name *"nr"* is reserved for referencing, by means of a number, *dimensionless* entities which we do not wish to consider to be numbers. The name "nr" may be used with more than one entity type, but the occurrences are expanded by prepending the name of the entity type (in lower case) and an underscore to produce unique predicate names (see Figure 5.6). If a numeric subtype definition (e.g. [1..7], {1,2}) is specified beside the entity type this is understood to specify the implicit numeric subtype. In Figure 5.6 we show this as [..]. If no numeric subtype definition is given, Real is assumed.

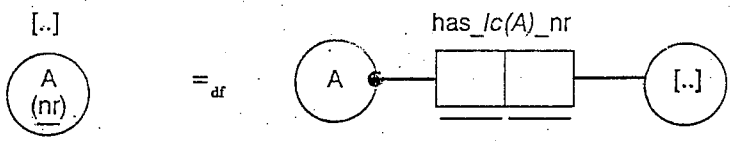


Figure 5.6 Abbreviating reference of a dimensionless entity

For example, "Quantity (nr)" and "Rating (nr)" generate the predicates "... has_quantity_nr ..." and "... has_rating_nr ...". If desired, a name for the numeric subtype may be constructed by prepending the entity type name to "_nr", e.g. "Rating_nr": if used this must be underlined. If we wish to treat a dimensionless quantity as a number then its name is underlined and no reference scheme is specified for it on the diagram.

We now examine *unit-based reference modes* (e.g. cm, $). Names of unit-based reference modes are unique, and never abbreviated. The reference predicate name is generated by prepending "... has_" to the reference mode name (not underlined), and then appending "_value ...". See Figure 5.7 (here we assume *r* is not "nr"). If no numeric subtype definition is specified, then Real is assumed. In the unlikely case where a name for the numeric subtype is desired, this may be constructed by prepending "Nr_for_" to the name of the reference mode (e.g. "Nr_for_Mm", "Nr_for_mm", "Nr_for_$").

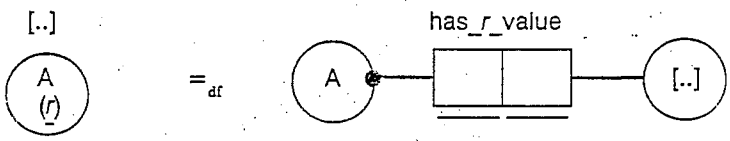


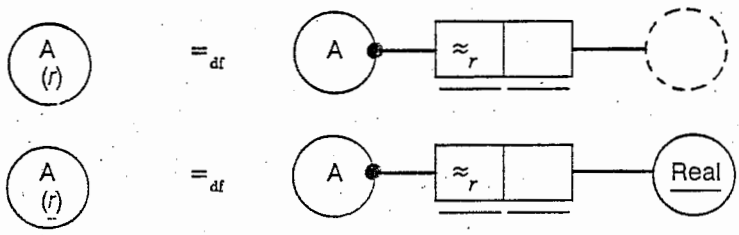
Figure 5.7 Abbreviating an injective unit-based reference scheme

For example, "Length (cm)" and "Money ($)" generate the predicates "... has_cm_value ..." and "... has_$_value ...". We use the term "unit-based"

liberally to include cases such as "Year (AD)" and "Portion (%)": these respectively generate the predicates "... has_AD_value ..." and "... has_%_value ...".

For convenience we now introduce a metasymbol to facilitate general discussion about simple injective reference schemes. Here "simple" means neither composite not disjunctive (see later). In doing so we summarize the simple reference translation rules discussed so far.

If a reference mode r is simple, its reference predicate may be abbreviated as "$\approx_r$". A generic reading for "$x \approx_r y$" is "x is identified (under r) by y"; however its specific reading is determined by the cited reference mode r in accordance with our earlier abbreviations. See Figure 5.8. Here "a" denotes the lowercase version of the name "A". For example, if "A (r)" is replaced by "Gender (code)" then "$x \approx_r y$" is replaced by "x has_gendercode y". Although "$\approx_r$" bears some analogy to the equality operator "=", clearly "$\approx_r$" is not reflexive, nor symmetric nor transitive.



r	$\approx_r$	
name	has_aname	{ a = lc(A) }
code	has_acode	{ a = lc(A) }
title	has_atitle	{ a = lc(A) }
r	has_r	{ $r \neq$ 'name','code','title' }
nr	has_a_nr	{ a = lc(A) }
r	has_r_value	{ $r \neq$ 'nr' }

Figure 5.8 Summary of simple reference predicate translations

The inclusion of a reference mode for an entity type A implies that objects of type A are Described, and hence falsifies any direct comparison between an object of type A and a number. For example, the specification "IQ (nr) [0..200]" makes false an assertion such as ∃x(IQ x & x=130); any comparisons with a number will have to be made *indirectly* via the numeric reference scheme. If IQ is instead specified as the subtype [0..200] then such assertions are not automatically rejected. This choice is made in Figure 5.9. The subtype definitions illustrate indirect and direct comparisons with numbers.

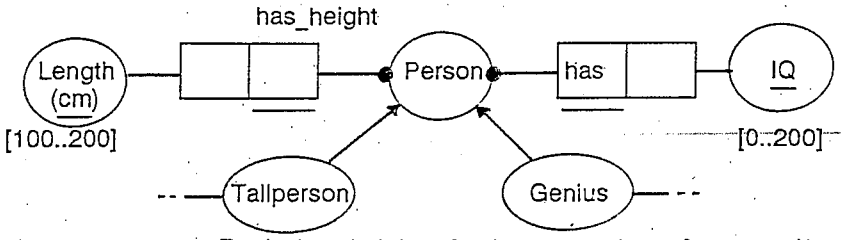


Figure 5.9 Indirect and direct comparisons with numbers

We do not specify a general notation for abbreviating composite reference schemes other than the simple lexical case considered earlier. In practice, designers may introduce their own notations for special cases. We suggest an asterisk then be used to indicate a fuller specification exists elsewhere. For example, "Date (*ymd)" might abbreviate a schema module in which Date is uniquely determined by Year (AD) [0..9999], Month (nr) [1..12] and Day (nr) [1..31] together with other constraints (e.g. month sizes), reference schemes (e.g. MonthName), and derivation rules for subtracting dates etc. In this approach a year is a single segment of the timeline, but months and days have multiple occurrences.

In rare cases, we may wish to allow an entity type to have more than one unit-based reference mode within the same information system. This may arise to cater for a gradual transition from an old to a new unit system (e.g. imperial to metric), or because even with the one entity type different units are used for different contexts (e.g. mm, m, km, pc etc.). Some background discussion is given in NH89 (pp. 169-70). We now discuss a slightly different scheme for formalizing such cases.

A conversion rule specifies how one unit may be converted to another and vice versa. Such a rule may be shown explicitly as a derived fact type and its textual form included in the list of derivation rules. Alternatively, the rule may be abbreviated in equation form and written beside the entity type ellipse. If *r* is the name of a unit-based reference mode for the entity type, *s* is the name of another unit, and *f*(x) is a function-term of KL then a *conversion equation* between these units may be specified and translated into KL as shown in Figure 5.10.

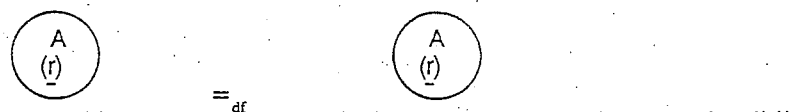


Figure 5.10 Defining alternative units for the same entity type

For example, consider the schema fragment shown in Figure 5.11. This reflects the practice in the Australian lumber industry, where cross sectional measurements are given in millimetres and the reach (long dimension) is measured in metres.

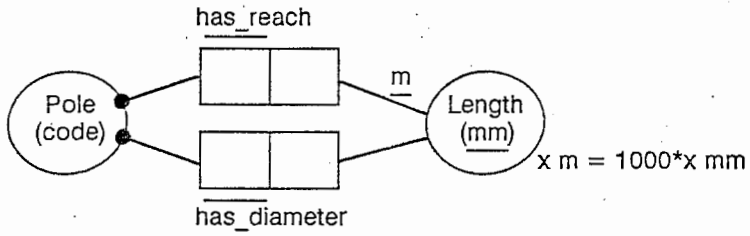


Figure 5.11 Length may be measured in mm or m

The conversion equation translates as:

$$\forall xy(\text{ x has\_m\_value y} \equiv \text{x has\_mm\_value 1000*y })$$

Notice the "m" beside the top right-hand role arc in Figure 5.11. This aspect cannot be captured conceptually, but has an implementation effect. The placing of "(mm)" and "m" indicate that mm is the primary reference mode for Length, but for input and output the reach is given in metres. The lack of any unit marker on the diameter role means that diameters are given in the primary unit (mm).

As an example, suppose we wanted to know which poles had a reach which was 50 times their diameter. Using "$\{x\colon \Phi x\}$" to denote the set of all x such that Φx, this query may be formulated in a primitive conceptual query language thus:

List {x: ∃yzwv(y has_polecode x & y has_diameter w & w has_mm_value z
& y has_reach v & v has_mm_value 50*z) }

If the schema is implemented with reach values in metres, the query phrase "v has_mm_value 50*z" calls the conversion rule to compute the mm value from the stored m value. Axioms introduced later in the chapter enable this query to be expressed more concisely.

Sometimes, it is useful to be able to specify a number of subtypes of the same supertype without having to draw a separate node for each. Let A be a described entity type which is numerically referenced by the reference mode r, where r is either nr or is unit-based, and let "[..]" denote any bracketed numeric subtype definition as specified earlier. We allow "[..]" to be

written beside a *role* played by *A* to specify that this role is played only by the subtype of *A* defined in terms of [..]. The general abbreviation scheme is set out in Figure 5.12.

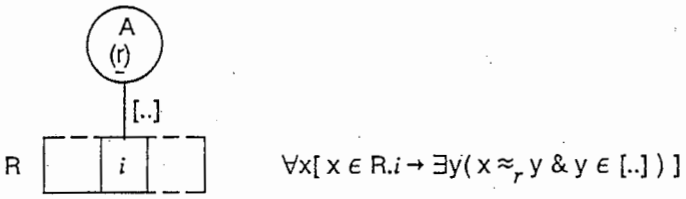
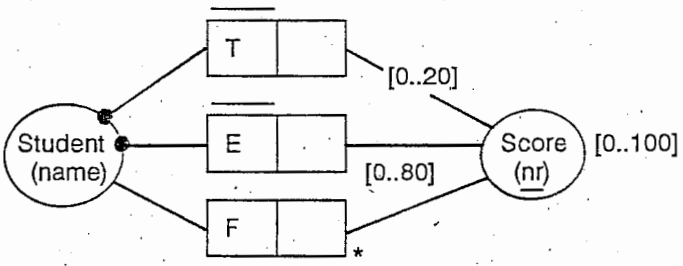


Figure 5.12 Concisely specifying a numerically referenced subtype

A simple example is given in Figure 5.13. Here the test is marked out of 20 and the exam out of 80. Notice that the numeric constraint on Score must cater for all the roles it plays (including the derived role). Without the abbreviation just introduced, it would be necessary to explicitly depict nodes for TestScore and ExamScore (with the former a subtype of the latter, which itself is a subtype of Score). If the schema is complete with respect to the roles played by Score, the [0..100] constraint may be omitted since it is then derivable.

Notice the use of a *predicate dictionary* to enable long predicate names to be abbreviated within the diagram and the derivation rule. With large schemas, all the predicates may be abbreviated as "R_1", "R_2" etc.



T = obtains a test score of
E = obtains an exam score of
F = obtains a final score of

* $\forall xy[\, xFy \equiv \exists y_1 y_2 (\, xTy_1 \,\&\, xEy_2 \,\&\, y = y_1 + y_2 \,)\,]$

Figure 5.13 Two uses of the shorthand notation of Figure 5.12

The specification of the derivation rule requires further discussion, since so far we have not axiomatized + as an operator between scores (which here are neither numbers nor strings). We discuss this later in the chapter.

If a different unit-based reference mode is also specified on a role arc with a [..] constraint, then this mode is used to generate the subtype definition. For example, with Figure 5.11, a [2..5] constraint next to "m" would specify that the reach of a pole must be 2, 3, 4 or 5 metres.

In rare cases, we may wish to allow an *xor reference scheme*, involving an exclusive disjunction of two 1:1 reference predicates. We allow such cases to be abbreviated by parenthesizing the names of the reference modes, separated by a *stroke* "|" (see Figure 5.14). Here r_1, r_2 may be lexical or numeric reference modes, and $\approx_{r1}$, $\approx_{r2}$ are their associated reference predicates.

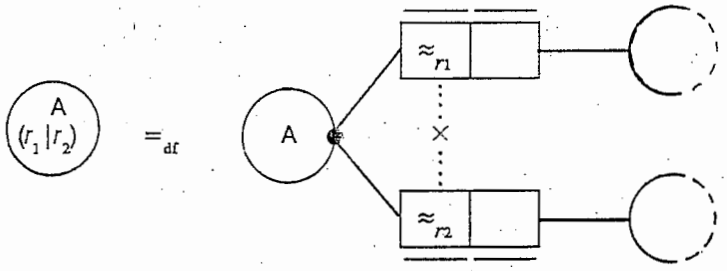


Figure 5.14 An xor reference scheme

For example, suppose our application deals with the last few thousand years and we wish to allow years to be denoted using the Christian convention (e.g. 500 BC, 1989 AD). This may be set out in short or long form as shown in Figure 5.15. The use of the "*" in "BC*" is explained later in the chapter.

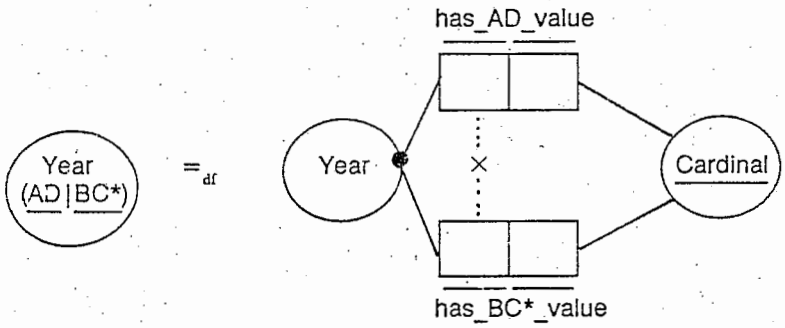


Figure 5.15 An example of an xor reference scheme

If the reference scheme of Figure 5.15 is used, then year entries shown in NIAM fact tables must include the "AD" or "BC" suffix. This suggests an alternative way of conceptualizing this reference scheme (see Figure 5.16). Equivalence between these schemas can be proved. However, we generally prefer the first approach because it simplifies the specification of +, - and ordering operations on years (see later) as well as separate numeric subtype constraints (e.g. [1..6000] for BC and [0..2000] for AD).

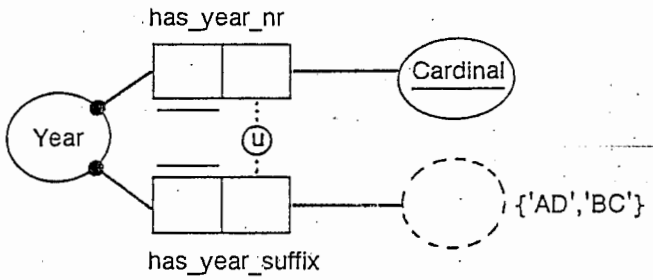


Figure 5.16 An alternative but usually inferior conceptualization

In a somewhat similar vein, one might conceive of unit-based reference in terms of a composite reference in which the unit is objectified. For example, Figure 5.17 might be proposed to deal with different units for Length. We prefer our earlier approach for the same reasons (easier to specify number-like operations and related numeric subtypes).

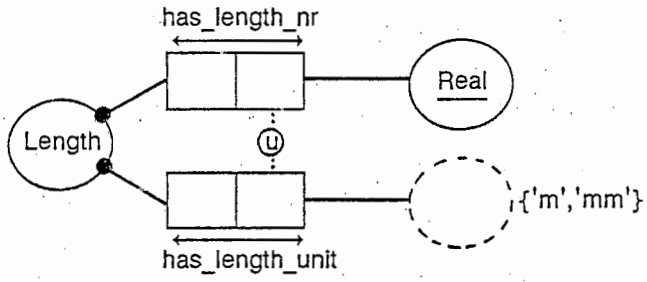


Figure 5.17 A generally inferior way to conceptualize lengths

Informally, a kind of subtyping scheme for all unit-based entities is sometimes proposed. For example, an entity type Unit_based_entity might be referenced by predicates to Real and Unit, with subtypes of Length, Mass etc. However, this approach conflicts with our general philosophy of allowing subtypes of the same supertype to be meaningfully compared, and it lacks the simple advantages cited earlier. Hence we reject this approach.

Role-object constraints (sec. 4.5) may be applied to described objects via their primary reference scheme. If A (r) {..,a,..} is depicted, where the reference mode r is either lexical or numeric with associated reference predicate $\approx_r$, and A plays role i of predicate R, then connecting this role arc via a broken arc to a has the following semantics when a is marked as shown:

$\overset{\bullet}{\{..,a,..\}}$ $\quad \exists x(x \in R.i) \rightarrow \exists y(y \in R.i \;\&\; y \approx_r a)$

$\{..,\underline{a},..\}$ $\quad \exists^{?}\underline{x}(R\underline{x} \;\&\; x_i \approx_r a)$

$\overset{n;m}{\{..,a,..\}}$ $\quad \exists^{n;m}\underline{x}(R\underline{x} \;\&\; x_i \approx_r a)$

5.2 Global aspects

So far all our mapping rules from NIAM notation to KL have been specified so that they can be carried out on diagram components, independently of the rest of the diagram. This incremental approach greatly simplifies the task of either manual or automated mapping. There are some aspects of NIAM diagrams however that must be interpreted on a *global* rather than local basis. In particular, we need to specify *mutual exclusion between primitive described object types*, and cater for *implicit mandatory roles*. These features are examined in this section.

Earlier we partitioned the domain into described objects, numbers, strings, pairs and {nil}. Any mutual exclusion between subtypes of String or subtypes of Real is implied by the subtype definitions. Any mutual exclusion between pair types is implied by exclusion between their corresponding component types or by other constraints, e.g. explicit exclusion constraints between their role sequences. In rare cases, pair types may be explicitly specified as subtypes of other pair types (e.g. different information might be recorded for (Person,Subject) pairs according to the subject): in such cases, exclusion between subtypes is determined from other constraints in the usual way. However, we have yet to consider mutual exclusion between described object types.

An object type is *primitive* iff it is not defined in terms of another object type, i.e. iff it is not a defined subtype. Recall (sec. 4.1) that a described object type appears as a solid ellipse, with no embedded roles, and with a name that is not underlined. Once the whole conceptual schema diagram is available, the described object types which are primitive may be identified: their ellipses are not the sources of any subtype arrows.

For any conceptual schema there will be a finite number of primitive described object types $A_1, .., A_n$. The following axioms are now obtained from the global CS to specify that the described objects are partitioned into these types:

P3 $\forall x$ (Described $x \rightarrow A_1x \vee .. \vee A_nx$)

P4 $\forall x$ [$\sim(A_1x \& A_2x) \& \sim(A_1x \& A_3x) \& .. \& \sim(A_{n-1}x \& A_nx)$]

This partition is portrayed in Figure 5.18, using the predicate names as type names. Each of these primitive types appears as a solid ellipse on the CS diagram, and each may have subtypes.

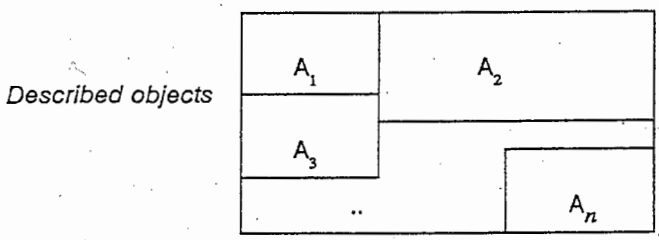


Figure 5.18 Partitioning the described objects into primitive types

Note that the mutual exclusion aspect of partitioning axioms renders false any assertion that objects in different primitive types are equal. For example, if Person and Department are primitive, the sentence ∃xy(Person x & Department y & x = y) is rejected. Moreover, the behaviour of other comparison operations (e.g. ≤) has not been specified between objects of mutually exclusive types. For example, we have axiomatized ≤ between real numbers and between strings, but not between real numbers and strings. Likewise special comparison operations to be defined for described objects will be relativized to the appropriate type, e.g. the predicate ".. is before or simultaneous with .." might be defined only for objects of type Date (further details on this matter are discussed in the next section).

Mutual exclusion between primitive object types has been axiomatized. If subtypes of the same primitive type are mutually exclusive, this is captured by the subtype definitions. The term "mutually exclusive" should not be taken to imply that "migration between types" is impossible. For example, consider the schema of Figure 5.19.

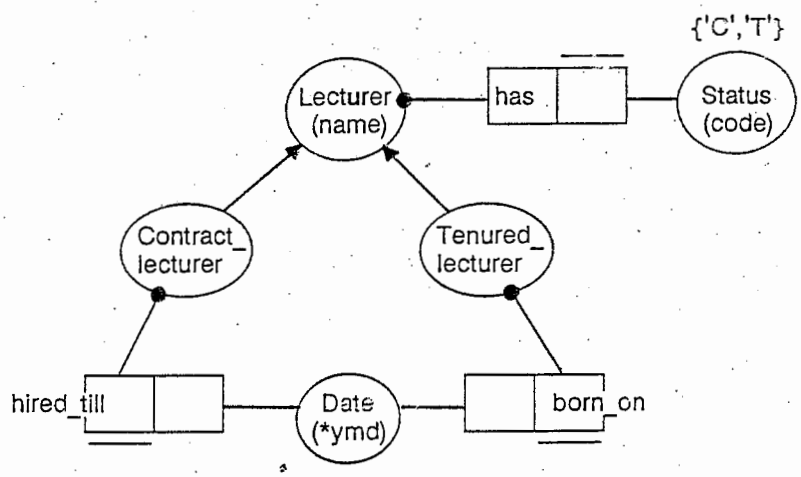


Figure 5.19 Migration between "exclusive subtypes" is allowed

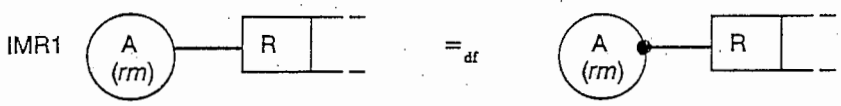
When this schema is mapped to KL it is easily proved that the subtypes Contract_lecturer and Tenured_lecturer are mutually exclusive. The key steps in the proof involve the uniqueness constraint on has_status, the subtype definitions, and the inequality "C" ≠ "T". Yet we wish to allow (thankfully!) that migration from Contract_lecturer to Tenured_lecturer is possible. A superficial analysis might suggest a contradiction here: how can the same object belong to two mutually exclusive types?

Briefly, the problem is solved by saying that although we may not assert that an object is simultaneously both a contracted and a tenured lecturer, we may assert that the object is a contracted lecturer at time t_1 and a tenured lecturer at time t_2, so long as $t_1 \neq t_2$. This is consistent with our earlier analysis (section 3.1), where each interpretation of a CS is a UoD subworld, and time-dependent sentences are indexed to their time of utterance.

We now consider a notation for specifying *MRC*s (mandatory role constraints) *implicitly*. This notion is discussed in our earlier work (NH89, sec. 6.3), but our present treatment is more refined, and expands the allowable range of conceptual schemas. A convention for implicit MRCs is useful for the following reasons: to enable theorems about modal relationships between subschemas to be expressed more concisely without loss of generality; to emphasize the more important MRCs; to simplify the drawing of CS diagrams (by reducing the number of dots, and often avoiding the need to connect disjunctive MRCs); and to encourage avoidance of lazy entities (see later).

Figure 5.19 has already made use of the implicit MRC notation; but before discussing this example we set out the general notation. First note that the implicit MRC notation may be used with an object type *A* if and only if both the following conditions are satisfied: (1) *all* the roles played by *A* in the *global* CS are specified; (2) a *reference mode* for *A* is cited in parentheses. One consequence of condition (2) is that the implicit MRC notation cannot be used with subtypes, pair types, numeric types, or string types. With our new approach, reference modes for pair types cannot be cited on a CS diagram.

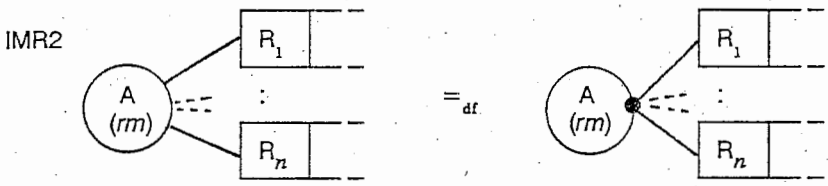
The simple case is set out in Figure 5.20. Here *A* is a described object type with reference mode *rm* which may be lexical, numeric, composite, disjunctive or defined (e.g. *ymd). If globally, in addition to its reference role(s), *A* plays only the role R.1, an MRC on this role is implied.



where globally there are no other roles played by A

Figure 5.20 A globally implied mandatory role constraint

The disjunctive case is set out in Figure 5.21. Here, apart from its reference role(s), *A* globally plays only roles $R_1.1$, .., $R_n.1$. It is implied that the disjunction of roles $R_1.1$, .., $R_n.1$ is mandatory.



where globally there are no other roles played by A

Figure 5.21 A globally implied disjunctive MRC

As indicated, we refer to these global abbreviation rules for implied MRCs as *IMR1* and *IMR2*. If Figure 5.19 includes all the roles played by Status, Lecturer and Date in the global schema, then IMR1 tells us that both the roles in the has_status predicate are mandatory. Although globally the dot on Lecturer could have been omitted since it is implied, we recommend including it to emphasize that the subtype defining role is mandatory. IMR2 tells us that, if Date plays no other roles globally, the disjunction of the two non-reference roles played by Date is mandatory.

Implicit MRC notation is not used with numeric or string object types, since we do not wish to assert MRCs for such types (which often include overlapping subtypes anyway). The notation is not used for subtypes or pair types, since we do not count it as remarkable for a role played by a subtype or pair type to be optional (for that type). Let us use the term *plain entity type* (PET) to mean a primitive, described object type. Parenthesized reference modes may be cited only for PETs. Hence IMR1 and IMR2 apply only to PETs.

It is very unusual to include in a schema any PET which does not play a mandatory role (or role disjunction) other than the roles needed to assert its existence. Indeed, in NH89 we effectively made it a CS formation rule that each PET had to play some role (or role disjunction) other than its reference role(s). The philosophy behind this approach was that there is little point

in positing an object unless it actually does something. For reasons given below, we no longer feel this should be a conceptual requirement, and have downgraded the status of this viewpoint from a CS formation rule to an implementation warning.

We call a described object which "does nothing" a *lazy entity* (i.e. the only thing known about a lazy entity is that it exists). An entity type which allows lazy instances is a *lazy entity type* (LET). To discourage the designer from allowing LETs except in exceptional cases, we make it a formation rule that LETs must have their reference schemes specified explicitly, i.e. these schemes must not be abbreviated by parenthesized reference modes. To take an extreme example, Figure 5.22 shows one legal but boring global conceptual schema diagram. Here Lecturer is a lazy entity type, since its only non-referential role is optional. In this UoD we allow that we know the name of some lecturers but not their status.

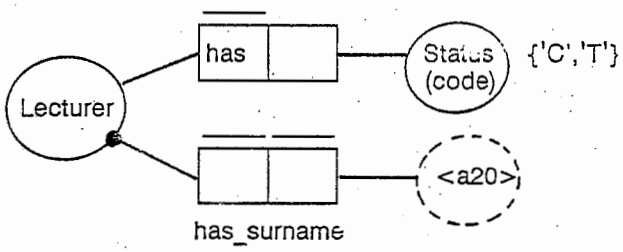


Figure 5.22 An unusual example where some lecturers are "lazy"

Note that it would be incorrect to try to abbreviate this global schema by parenthesizing the reference mode of Lecturer. If we did this, the has_status role would be implicitly mandatory (cf. Figure 5.19). There are two main reasons for allowing the possibility of lazy entities. Firstly, they are sometimes needed in practical applications (e.g. we might only know the surname of a lecturer and still want to record this information): attempts to handle such situations without lazy entities by adding a unary predicate "exists" suffer the logical problems of trying to treat existence as a predicate. Secondly, the formal inclusion of LETs enables a more uniform treatment of conceptual schema transformations.

We define an *active entity* to be an entity which plays at least some role other than its reference role(s). So an entity is active if and only if it is not lazy. Typically, a lazy entity type may include some active entities, e.g. Lecturer in Figure 5.22 may be instantiated by objects who also instantiate the has_status role. Though of little practical significance, we allow the possibility of described object types with no active entities. Such

entity types are called *completely lazy entity types* (CLETs). An extreme example is obtained by removing the has_status predicate and the Status object type from the UoD of Figure 5.22. If has_surname is the only role globally played by Lecturer, then Lecturer is a CLET. Note that while subtypes and pair types may be LETs they may never be CLETs. Moreover, even CLETs must have a mandatory 1:1 reference scheme.

We conclude this section by adding a further graphic notation which simplifies the drawing of large schemas. Sometimes, an object type (e.g., Date, Money) may play so many roles in the global schema that it is awkward to connect all the relevant role arcs to the ellipse for this object type. In this situation we allow, as an alternative, that the object type may be displayed several times, using a *double ellipse*. Optionally, the notation "*n* of *m*" may be added to indicate this is the *n*th occurrence of the object type out of a total of *m* occurrences (see Figure 5.23).

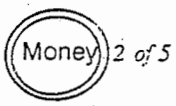


Figure 5.23 Double ellipse notation allows multiple occurrences

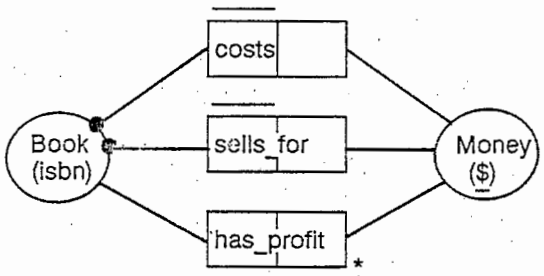
In this case any mandatory role dot is interpreted as applying to the whole object type. However, before applying the globally implied mandatory role constraint rules (IMR1-2), all the roles collectively played by all the occurrences of the object type need to be included.

5.3 Derivation rules

Besides fact types and constraints, a conceptual schema may include *derivation rules*. In this section we illustrate how derivation rules may be formulated in KL. The +, - and ≤ operators are axiomatized for most numerically referenced entities, and used to facilitate the specification of derivation rules and textual constraints. Finally we contrast subset and equality constraints with derivation rules.

Basically, a derivation rule is a sentence which *defines*, at least partly, a predicate or a function. For example, we might define the derived predicate father_of in terms of the predicates parent_of and has_gender. Although in some contexts the term "derivation rule" is used as an alias for "inference rule", we do not treat these terms as synonyms. Modus Ponens is an inference rule (or transformation rule) but not a derivation rule.

Derived predicates and functions are often depicted graphically in terms of asterisked box-chains. In any case, they must be specified textually as derivation rules, and any lexical or numeric constraints on associated object types must still permit the derived values (recall Figure 5.13). In our formalization, a derivation rule must take the form of a universally quantified biconditional or a universally quantified conditional. In practice, *biconditionals* are typically far more common, so we examine these first. Consider Figure 5.24. This might be part of a schema used by a book retailer. Here the profit on a book is derived by subtracting its cost price from its retail price. *Braces* may be used with derivation rules to enclose *comments*: these are intended only for humans, and are ignored when mapping to KL.



* { profit = sales price - cost price }
∀xy[x has_profit y ≡ ∃zw(x costs z & x sells_for w & y = w-z)]

Figure 5.24 Comments may be included in braces

Notice the term "w-z". We have not yet axiomatized - for Money objects. Our earlier example of adding scores (Figure 5.13) is similar. In both cases we have a *numerically referenced entity type* (here, Money and Score) for which we found it convenient to use + or - as operators for *addition or subtraction* on this type. We reserve the term *NRE* (numerically referenced entity) for objects belonging to a type with an injective reference link to Real. To facilitate the formulation of derivation rules involving NRE types we now axiomatize +, - and ≤ for such types.

Let *A* be a primitive NRE type which is numerically referenced by the reference mode *r*, which is either nr or is unit-based (but not asterisked: see later). Let $\approx_r$ denote the associated reference predicate (has_*lc(A)*_nr or has_*r*_value). Then for each specific CS, the following axioms are added for each such A included in it.

NRE÷ $\forall xyz[\, Ax \,\&\, Ay \,\&\, Az \,\&\, z = x+y \equiv$
$\exists x_1 y_1 z_1 (\, x \approx_r x_1 \,\&\, y \approx_r y_1 \,\&\, z \approx_r z_1 \,\&\, z_1 = x_1 + y_1 \,)]$

NRE- $\forall xyz[\, Ax \,\&\, Ay \,\&\, Az \,\&\, z = x-y \equiv$
$\exists x_1 y_1 z_1 (\, x \approx_r x_1 \,\&\, y \approx_r y_1 \,\&\, z \approx_r z_1 \,\&\, z_1 = x_1 - y_1 \,)]$

NRE≤ $\forall xy[\, Ax \,\&\, Ay \rightarrow (\, x \leq y \equiv$
$\exists x_1 y_1 (\, x \approx_r x_1 \,\&\, y \approx_r y_1 \,\&\, x_1 \leq y_1 \,))]$

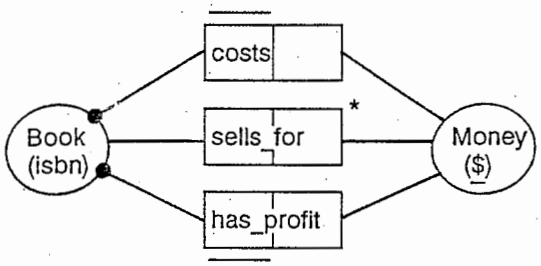
Apart from exceptional cases to be discussed later, these axioms define addition, subtraction and ordering of numerically referenced entities of the same type in terms of adding, subtracting or ordering the real numbers with which they are in 1:1 correspondence via the reference predicate. This predicate may be depicted implicitly with a reference mode, or explicitly. If the reference mode notation is not used, and more than one injective reference to Real is specified, then we pick the reference predicate whose name is alphabetically prior. Each specific CS will have its own specific NRE+, NRE- and NRE≤ axiom lists.

Although the Reals are closed under + and -, the same cannot generally be said for NRE types, e.g. 100°C and 500°C are temperatures but -400°C is not (at least according to current physics). We take the referent of such out-of-range expressions to be nil. Note that with our untyped calculus, any term of the form x+y will refer once x and y have been assigned a value; so the main aspect of NRE+ is to specify when A x+y. Note also that $A\ f(a_1,..,a_n)$ is equivalent to $\exists x[x=f(a_1,..,a_n) \,\&\, Ax]$. Given the other constraints on the referential predicate, the only extra information gained from using "≡" instead of "→" in axiom NRE+ is that when the right operand of "≡" is true, z = x+y (this rules out the possibility that $\sim A$ x+y). A similar comment applies to NRE-.

From NRE+ and the real field axioms RF1-2 it follows that addition of numerically referenced entities is commutative and associative. We have already specified that ≤ is transitive and antisymmetric for any type (axioms TO1-2). From NRE≤ and RTO3 it follows that ≤ provides a total order for any non-asterisked NRE type. We do not define +, - or ≤ for described entities that are not NREs. Neither do we define + or - or ≤ between different primitive NRE types, or between these types and numbers or strings. Moreover, we do not generally define other numeric operations such as * and / for NREs since these will not always be meaningful (e.g. multiplying two temperatures) and the result of such an operation may not be of the same type (e.g. length * length ⟶ area).

In the next chapter, we develop the notion of equivalence between conceptual schemas in terms of logical equivalence. So conceptually there are

a number of equivalent ways of specifying biconditionality dependencies (cf. degrees of freedom in an equation). For example, consider Figure 5.25. This is like Figure 5.24, except the sale price is derived by adding the profit to the cost price, and UCs and MRCs are given for has_profit but not for sells_for.

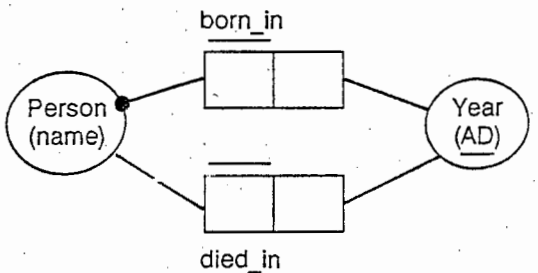


* { sales price = cost price + profit }
$\forall xy[\, x \text{ sells\_for } y \equiv \exists zw(\, x \text{ costs } z \,\&\, x \text{ has\_profit } w \,\&\, y = z+w\,)]$

Figure 5.25 This describes the same UoD as Figure 5.24

The equivalence between the subschemas in Figures 5.24 and 5.25 may be demonstrated with a deduction tree. Although logically equivalent, these schemas lead to different implementations, e.g. profit facts are derived in one but stored in the other.

So far all the constraints we have examined have been depicted diagrammatically: we call these *graphic constraints*. Other constraints may be specified by means of (possibly abbreviated) KL formulae : we call these *textual constraints* (TCs). These may be written below the diagram, with explanatory comments in braces. The axiomatization of +, - and ≤ for NREs often simplifies the formulation of textual constraints. Figure 5.26 provides one example.

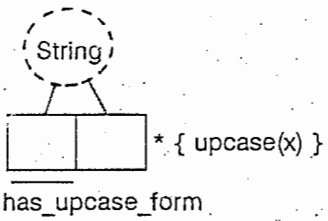


TC1: { one's birthyr occurs on or before one's deathyr }
$\forall xy_1y_2(\, x \text{ born\_in } y_1 \,\&\, x \text{ died\_in } y_2 \rightarrow y_1 \leq y_2\,)$

Figure 5.26 A textual constraint

Now suppose that instead of using AD as the reference mode for Year we used BC. The constraint TC1 would no longer be correct, since with the BC scheme the smaller the number the later the year (e.g. 100 BC is earlier than 50 BC). Clearly, here is a case where the NRE+-≤ axioms do not apply. Another example is the standard astronomical method of measuring absolute magnitude (the more negative the number the brighter the star). Such exceptions were foreshadowed. To prevent the NRE axioms being incorrectly applied to such unusual reference schemes, we demand that the names of such reference modes or reference predicates be asterisked (e.g. "BC*"). Recall that asterisked reference schemes were excluded from the NRE+-≤ axiom lists. We leave it to the designer to specify specialized axioms for such rare, exceptional cases if this is desired. For example, the NRE≤ axiom could be adapted for BC* by replacing "$x_1 \leq y_1$" with "$x_1 \geq y_1$".

Though it is never necessary, it is often convenient to express some derivation rules in functional notation rather than defining an equivalent predicate formulation. As an example, which also illustrates that it is useful to allow relationships between lexical objects, consider Figure 5.27. Here *upcase* is a *derived function*, which accepts a string argument and returns the string with all its lower-case letters shifted into upper-case. For example, upcase('phd') = 'PHD' and upcase('PhD') = 'PHD'. Though such functions may be depicted graphically as predicates using asterisked braces as shown, usually they are omitted from the CS diagram.

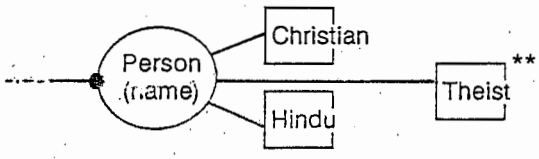



```
* { upcase(x) }
∀xy[ Char x & Char y → (y = upcase(x) ≡
    ord(x) ∈ [97..122] & ord(y) = ord(x)-32 ∨ ord(x) ∉ [97..122] & x = y )]
∀xy[ String x & String y → (y = upcase(x) ≡
    head(y) = upcase(head(x)) & rest(y) = upcase(rest(x)) )]
```

Figure 5.27 A derived predicate expressed in functional notation

In rare cases, derivation rules may take the form of a *conditional* rather than a biconditional. As a simple example, consider the subschema of Figure 5.28. There are several things to note about this example. First, to enable the predicate being defined to be written first, we introduce "**if**" as the propositional operator for "is materially implied by", i.e. given any wffs α

and ß, α **if** ß $=_{df}$ $\alpha \rightarrow$ ß. We also allow "←" as a variant for "**if**", and "**iff**" as a variant for "≡".



** ∀x(Theist x **if** Christian x ∨ Hindu x)

Figure 5.28 Conditional derivation rules are marked "**"

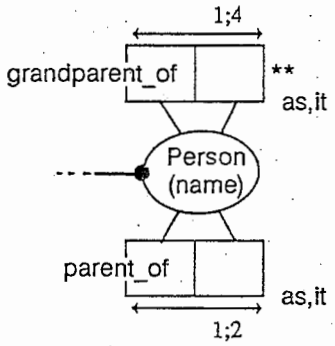
Note that *two asterisks* are used instead of the usual one. We make it a CS formation rule that biconditional derivation rules be singly asterisked, and conditional derivation rules be doubly asterisked. This new notation suggests that there is more being specified than just a simple derivation rule. Placing two asterisks besides a predicate indicates that the derivation rule provides only a *partial definition* of the predicate. This is reflected in the use of "**if**" rather than "**iff**".

In this UoD it is possible that some people are known to be theists without being known to be Christian or Hindu (e.g. they might be Jewish, or we might simply not know what their religion is). With this schema, it is consistent to assert that a person is a theist without asserting that he or she is a Christian or Hindu. From the implementation viewpoint, this means that doubly asterisked predicates may be partly derived and partly stored.

On a CS diagram no predicate (asterisked or not) may appear more than once. However, when a derivation rule is specified textually, more than one formula may be used (e.g. upcase(x)): in this case the derivation rule is the conjunction of these formulae. If the predicate is singly or doubly asterisked, each such formula must be a biconditional or conditional respectively. Unlike some systems (e.g. closed world Prolog), we do not treat the conjunction of conditionals as implicitly a biconditional. For example, if the derivation rule for Theist is ∀x(Theist x **if** Christian x) & ∀x(Theist x **if** Hindu x), this is not to be interpreted as ∀x(Theist x **iff** Christian x ∨ Hindu x). If the latter interpretation is intended, the predicate must be singly asterisked and defined as this biconditional.

Any doubly asterisked predicate must be included on the CS diagram with all its constraints marked. Moreover, *these constraints must be translated into KL* rather than being ignored, since the derivation rule specifies only part of the predicate. From the implementation viewpoint, for any state of the knowledge base the constraints apply to the combined population of stored

and derived instances of the predicate. Since the stored instances can affect the constraint satisfaction of this population, the constraints are not implied by a correctly formulated derivation rule (unlike the biconditional case). For example, consider Figure 5.29.



** ∀xz[x grandparent_of z **if** ∃y(x parent_of y & y parent_of z)]

Figure 5.29 Constraints on stored part of ** are not implied

The grandparent_of predicate is often used in Prolog texts as a paradigm case of a partly stored and partly derived predicate. Besides the grandparenthood information derived from parenthood facts, this approach allows the assertion of grandparenthood facts for those cases where we might not know who any of the intermediate parents are. The population of grandparent_of (including these stored facts) must obey the constraints.

Note that our double asterisk notation replaces our earlier notation (NH89, sec. 9.3) where we allowed a predicate to be shown twice on a diagram (once for stored and once for derived). The old notation is no longer permitted. Our new approach leads to simpler diagrams, and to derivation rules being clearly specified as biconditionals (*) apart from exceptional cases (**).

In some, but not all, cases the same feature of a UoD may be captured by either a biconditional/conditional derivation rule or an equality/subset constraint. We now identify and provide design guidelines for these cases, beginning with equality constraints. The general case is set out in Figure 5.30. We describe this by saying that at least one of the operands of the equality constraint is a *whole predicate*. Here, the whole predicate S is one of the operands.

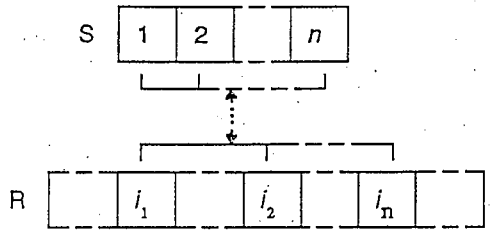
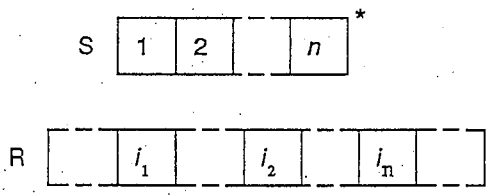


Figure 5.30 An equality constraint with a whole predicate operand

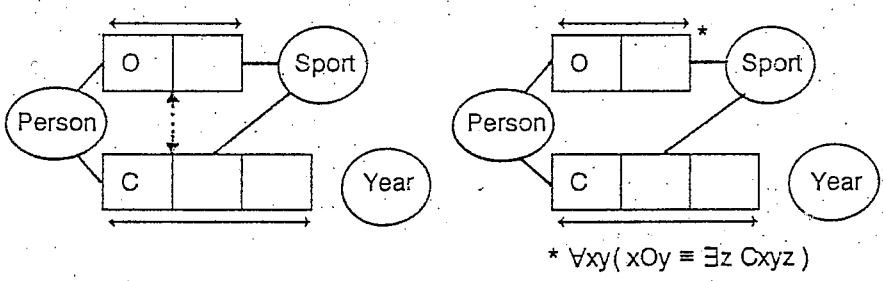
The equality constraint translates into the quantified biconditional formula shown in Figure 5.31, where this UoD feature has now been set out as a derivation rule. We generally recommend the derivation rule version since its implementation avoids the need to store the predicate S.



* $\forall x_1..x_n[\, Sx_1..x_n \equiv \exists \underline{z}(R\underline{z} \,\&\, x_1 = z_{i1} \,\&..\&\, x_n = z_{in})\,]$

Figure 5.31 Equivalent and usually preferable to diagram 5.30

For example, in Figure 5.32 the right hand version is usually preferred. In this UoD we cannot know that a person was an Olympian in a particular sport unless we know one of the years in which this occurred.



O = .. was_Olympian_in ..
C = .. *competed in .. in Olympic Games of ..*

Figure 5.32 The right hand version is usually preferred

In the rare cases where both operands of an equality constraint are whole predicates, either predicate may be selected as derived. For example, suppose people own a car if and only if they drive it. If it is desired to use both predicates Owns and Drives, one could be stored and updated while the other is derived from ∀xy(x owns y ≡ x drives y).

If neither operand of an equality constraint is a whole predicate, then we cannot recast this as a derivation rule. For example, if we want to know a person's resting heart rate if and only if we also want to know their reaction time then we specify this by means of an equality constraint between the first roles of has_resting_heart_rate and has_reaction_time (see NH89, p. 172). We cannot specify a derivation rule to enable a person's reaction time to be determined from their resting heart rate.

Let us now consider subset constraints. First note that, being uni-directional, a subset constraint cannot express a biconditionality. In situations where an **iff** is required, we usually specify a derivation rule (or a subtype if subtype-specific knowledge is required). For example, suppose we want to know whether a person is a genius, and this can be precisely defined by the person's IQ. If there is something we wish to know only about geniuses then we create a subtype for Genius with this specific role attached (recall Figure 5.9). However if there is no such specific role we simply specify a derivation rule for Genius: this will be identical to the formulation of a subtype definition for Genius, e.g. $\forall x$[Genius x $\equiv \exists y$(x has_iq y & y $\geq$ 150)]. We cannot handle this case with a subset constraint.

With a subset constraint, the role sequence which is the operand pointed to by the subset arrow is called the *target operand*. In Figure 5.33 the target operand of a subset constraint is a whole predicate.

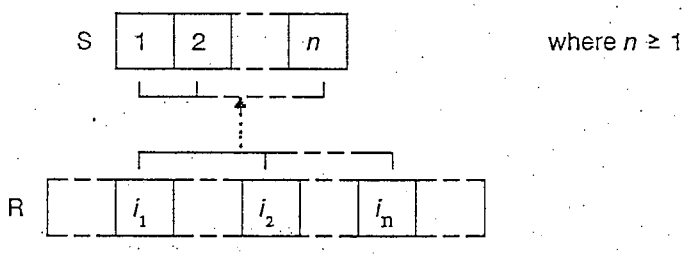


Figure 5.33 The target operand is a whole predicate

This subset constraint translates into the formula shown as a conditional derivation rule in Figure 5.34. Although conceptually, both figures specify the same UoD, the subset constraint version is usually preferable from the point of view of implementation.

One example is our Olympian schema, modified by replacing the equality constraint with a subset constraint (so we can know a person was an Olympian in a sport without knowing when this happened).

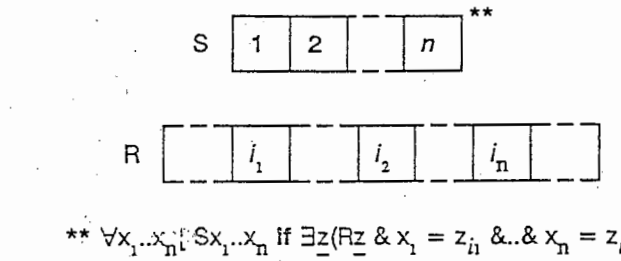


Figure 5.34 Equivalent to but often less preferable than 5.33

A simpler example, based on a case we discussed in NH89 (p. 206), is given in Figure 5.35. In this UoD if a person drives a car then that person owns that car. The subset constraint approach is adopted in the left-hand schema: here both the operands of the subset constraint are whole predicates. The subset constraint may be translated into KL as the formula used to specify the derivation rule in the right-hand schema. Another alternative is to attach Drives as a unary to (Person,Car) pairs in the Owns relation.

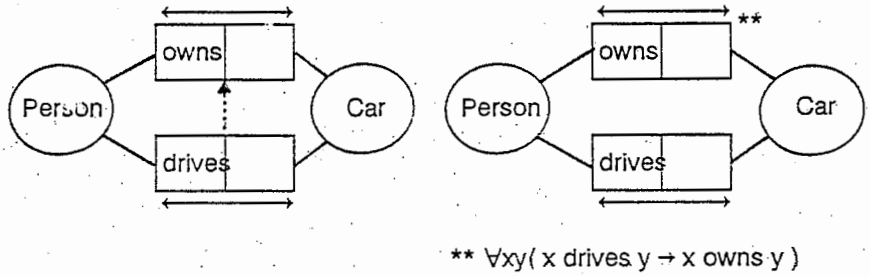


Figure 5.35 Both schema diagrams portray the same UoD feature

In NH89 we treated constraints as applying to the (stored) database. We now treat constraints as applying to the UoD and hence the knowledge base (what is known rather than what is stored). Which facts are stored and which are derived is not a conceptual issue: it makes no difference to the UoD. Hence to capture the UoD feature that people own any car they drive, the designer may choose either of the schema diagrams shown in Figure 5.35. Both map to the same set of KL formulae (though we also copy the asterisks over with derivation rules as an implementation directive to be acted on later).

When it comes to implementing a conceptual schema, we must make a decision as to what is to be stored or derived. At this implementation level we treat the diagrams differently. Consider the binary Olympian fact type in Figure 5.32. With the equality constraint, Olympian facts are actually stored and the constraint is enforced on relevant updates. With the derivation rule approach, the binary Olympian facts are not stored but are derived when

relevant queries are issued. For further discussion on various ways of implementing the two approaches with the owner-driver example, see NH89 (pp. 206-211, but recall that the diagram notation used there has been superceded by our new approach).

In NH89 (pp. 184-90) we classified relational properties as *positive* or *negative* according to whether the property could be used to deduce other (positive) facts or the negations of such facts. For example, reflexivity, symmetry and transitivity are positive whereas irreflexivity, asymmetry and intransitivity are negative. We suggested that negative properties be implemented as database constraints, but described how positive properties could be implemented as database constraints or conditional derivation rules.

The subset case that we have just examined fits within the general category of positive relational properties (though here two relations are involved). As a further implementation choice, it is always possible to replace a partially derived predicate by a stored predicate (used for updates) and a differently named fully derived predicate (used for queries), e.g. see our Synonym example (NH89 p. 186).

The general issue of whether to implement a feature by means of a database constraint or a derivation rule has been addressed by a number of researchers. A good overview of this research is provided by Gallaire, Minker and Nicolas (1984, pp. 173-5). We feel that our analysis has shed further light on this issue, by focussing on the nature of the derivation rule (biconditional vs conditional), identifying the cases where the design choice exists, and providing design guidelines for these cases. For NIAM in particular, our analysis adds a further class of conceptual equivalence theorems, which are formally provable in our system.

5.4 The database and definite descriptions

Recall that a knowledge base consists of a conceptual schema and a *database*. We have seen how a NIAM conceptual schema may be specified in KL. In this section we examine how the contents of a database state may be specified, and provide a more detailed account of *definite descriptions*.

Typically, a conceptual database state is a set of elementary facts. To begin with, an *elementary fact* is an assertion that one or more objects instantiate some predicate. Thus any elementary fact may be expressed in the

form $Ro_1..o_n$ where R is an n-ary predicate ($n \geq 1$) and o_1, .., o_n are *object designators* (i.e. each identifies an object in the domain of discourse). Moreover, an elementary fact cannot be rephrased as a conjunction of smaller facts without loss of information.

If a significant fact population is provided, elementariness can be verified by the projection-join check (see NH89 sec. 5.3); however the claim of significance relies on the intuitive understanding of the UoD expert who makes the claim. On the other hand, simple checks are available to show some classes of fact types are *not* elementary (e.g. if two roles are not spanned by a UC then the fact type is not elementary; and a predicate used for a pair type must have a full length UC (NH89, sec. 5.2).

Since elementarity has been discussed at length in NH89, and is not in general formally decidable, we do not dwell on the notion here. Instead, we address ourselves to the syntactical question of what is acceptable as a candidate elementary fact. We begin with a simple example (see Figure 5.36). One advantage of NIAM is that populations of fact-types (stored or derived) may be conveniently displayed in *fact-tables* beside the fact-types. A *knowledge-base diagram* (KB diagram) depicts a conceptual subschema as well as a sample population. In NH89, KB diagrams were called "schema-base diagrams".

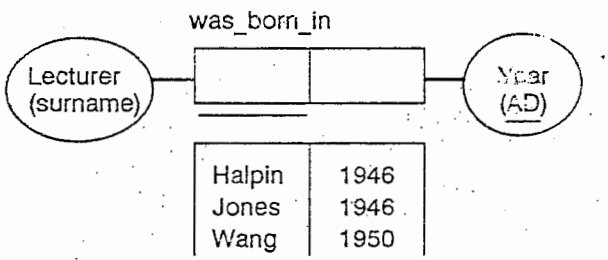


Figure 5.36 A simple knowledge-base diagram

The first row of the fact-table in Figure 5.36 is read as the following elementary fact:

F1: The Lecturer with surname 'Halpin' was born in the Year 1946 AD.

Such high level fact readings are determined by the following *Read Fact Algorithm* (RFA).

RFA:

Predicates are unabbreviated where relevant (see previous chapter);
Predicates are written with spaces for underscores;
Table entries are assigned their roles (predicate places) by position;
Each table entry e is associated with its object type A;
If A is lexical then $e \longrightarrow$ 'e';
If A is numeric or the reference mode is nr then e is unchanged;
In all other unnested cases a reference mode r is supplied for A
and table entries are rephrased as follows
A (r) $e \longrightarrow$ the A with r 'e'
A (r) $e \longrightarrow$ the A e r
A (r_1,...,r_n) $\longrightarrow$ the A with r_1 'e_1' and .. and r_n 'e_n';
If A is a pair type, then if the entry is (e_1,...,e_n) this is rephrased
as (e_1',...,e_n') { where the e_i' are as stated in RFA (recursive) }
The rephrased entries are written in place;
The first letter of the result is capitalized, and a period appended.

F1 may be viewed as the infix elementary fact $o_1 R o_2$, where R = ... was_born_in ..., o_1 = the Lecturer with surname 'Halpin', and o_2 = the Year 1946 AD. Here the object designators are *definite descriptions*: this is always the case for described entities. RFA indicates how such descriptions may be generated from the table entries (here: "Halpin", "1946") in the context of the schema. If the objects are strings or numbers, they are designated by string "constants" or numeric "constants".

This use of definite descriptions is a departure from conventional formalizations of databases, which unrealistically adopt "unique name axioms" in which all individuals are denoted by unique, unstructured, proper names (e.g. see Gallaire, Minker & Nicolas 1984, p. 160; Lundberg 1983, p. 91). Note also that the same object may be referenced by different descriptions (e.g. "the subject with subjectcode 'CS112'" and "the subject with subjecttitle 'Introduction to Information Systems'" may refer to the same object).

As discussed earlier, all our reference schemes are injective. When KB diagrams are used, we demand that a parenthesized reference mode (simple, composite or xor) be cited for each plain entity type: this provides its primary reference scheme for implementation. For a given knowledge base state, any definite description generated by RFA refers to exactly one object. For our static analysis these definite descriptions may be regarded as "structured individual constants".

If comparisons between objects are made across states, then the question arises as to whether these definite descriptions provide rigid designators

(i.e. must they refer to the same object in all states?). Clearly, this will usually be the intent (e.g. "the Year 1946 AD"). If flexible designators are permitted then these must be qualified before object comparisons between states are made (e.g. "the Lecturer with surname 'Halpin' at time t_1").

Our static analysis of reference schemes does permit the same string or number to be used within descriptions of different objects. For example, "Brisbane" may be used as a surname and a cityname within the same application. Of course the object referred to as "the Person with surname 'Brisbane'" cannot be the same as the object referred to as "the City with cityname 'Brisbane'", since our specific partition axioms would include an assertion that Person and City are disjoint object types. Nevertheless it is both meaningful and simple in our formalization to make indirect connections between objects of disjoint types via the strings or numbers used in their descriptions. For example, allowing any single lower-case letter as an IV in our primitive query language:

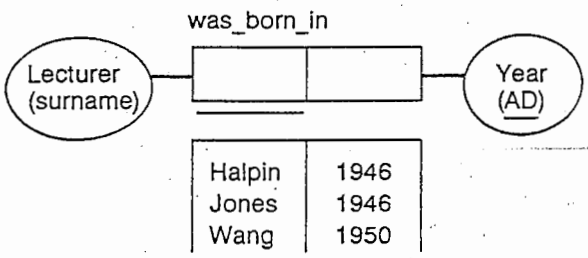
List the surnames of those who live in a city with the same name.
List {*n*: ∃*pc*(*p* has_surname *n* & *p* lives_in *c* & *c* has_cityname *n*)}

List the surnames of anyone whose mass in kg is numerically greater than his/her height in cm.
List {*n*: ∃*pmxh*(*p* has_surname *n* & *p* weighs *m* & *m* has_kg_value *x*
& *p* has_height *h* & *h* has_cm_value *x*)}

Although RFA provides a convenient fact formulation for human readability and communication, it is not actually used when the fact table is translated into KL. We now explain how the fact entries are translated, and clarify the theory of definite descriptions we are using. A KB diagram is translated in two stages: first the schema is translated into KL; then the fact entries are translated. From our earlier work, and ignoring the generic CS axioms, the schema may be translated as in Figure 5.37. If the schema were global, further axioms would be needed (e.g. implied MRCs).

Notice how the fact entries have been translated. Instead of a definite description formulation, simple existential formulae have been used. For example, fact f1 may be read as follows (compare this with F1):

Some object with surname 'Halpin' was born in an object with AD value 1946.



schema:

1 ∀x(Lecturer x ∨ Year x → Described x)
2 ∀x ~(Lecturer x & Year x)
3 ∀xy(x was_born_in y → Lecturer x & Year y)
4 ∀xyz(x was_born_in y & x was_born_in z → y = z)
5 ∀xy(x has_surname y → Lecturer x & String y)
6 ∀x(Lecturer x → ∃!y x has_surname y)
7 ∀y∃?x x has_surname y
8 ∀xy(x has_AD_value y → Year x & Real y)
9 ∀x(Year x → ∃!y x has_AD_value y)
10 ∀y∃?x x has_AD_value y
11 NRE+-≤ for AD

fact table:

f1 ∃xy(x has_surname 'Halpin' & y has_AD_value 1946 & x was_born_in y)
f2 ∃xy(x has_surname 'Jones' & y has_AD_value 1946 & x was_born_in y)
f3 ∃xy(x has_surname 'Wang' & y has_AD_value 1950 & x was_born_in y)

Figure 5.37 Translating a KB diagram

The reading cited for fact f1 is all that is required, since the schema (lines 3,5-10) provides the required context to ensure that there is exactly one object with surname 'Halpin' and that this object is a Lecturer, and that there is exactly one object which has AD value 1946 and that this object is a Year. This existential translation, in the context of our schema translation, provides the precise semantics for the definite description fact readings syntactically generated by RFA. This analysis generalizes straightforwardly to handle any *n*-ary elementary fact.

Our account of definite descriptions entails that such descriptions always succeed in referring (just as individual constants and ground function terms do). Earlier we included nil as a referent for garbage terms (e.g. "a"+2). In some analyses (e.g. ISO 1982, p. F-13) a similar referent is proposed for "impossible objects" e.g. square circles. With our approach, no described object can be nil. We place the onus on the designer to refrain from introducing impossible predicates; since the system has no formal means of detecting such absurdities, if they are used they will be taken to refer (e.g., "the Squarecircle with squarecirclename 'A'"). In practice, if such an

unlikely predicate is introduced, to keep our formal system compatible with possible world semantics we can always interpret the predicate in such a way as to make it possible.

Of more relevance is the possibility of definite descriptions which do not refer to real world objects, e.g. "the President with surname 'Raygun'". Again, we demand that such descriptions do refer. Recall our portrayal of a UoD as a set of possible subworlds. In some possible worlds a president with this surname does exist. Our analysis of singular terms (including definite descriptions) ensures that there is at least one possible subworld in which such terms do refer. This is all we need. While the conceptual information processor can ensure that the database is consistent with the CS supplied to it, it cannot ensure that the database is factual.

The treatment of definite descriptions is important, since our underlying formal proof mechanism (deduction trees) relies on successful reference. If instead we had adopted the Russellian theory of definite descriptions, and generated a branch formula such as $(\iota x)Ax \neq (\iota x)Ax$, this is merely equivalent to $\sim\exists!xAx$ and does not imply closure (e.g. see Rennie & Girle 1973, p. 215). Hilbert's analysis of ϵ could be used, since it guarantees reference and hence closure in this case. With our approach, no special operators for definite description are needed, and the non-identity of any ground term implies closure. A useful discussion of philosophical issues concerning singular terms is provided by Haack (1978, pp. 56-73).

Our earlier example of a KB diagram is typical, in that the ellipses depicted described object types. However, some or all of the ellipses may depict numeric or string types. Since our treatment of these possibilities differs from conventional NIAM, we discuss a few examples. Relationships between described and numeric objects must surely be admitted (e.g. F2-3), as well as relationships between described and lexical objects (e.g. F4-5).

F2: Person (surname) 'Jones' knows_the_kanji_for 3.
F3: Person (surname) 'Jones' has_IQ 120.
F4: Employee (emp#) '12345' has_employee_name 'Jones, E'.
F5: Person (surname) 'Jones' cannot_spell 'conceptualization'.

In the approach of Falkenberg (1986), any relationship between a non-lexical object and a lexical object is classified as a reference rather than as a fact. However, we feel that it is more natural to treat examples like F4-5 as facts rather than as references.

In principle, we permit relationships between numbers, though in practice these would tend to be derived rather than stored, e.g. 45 has_factor 5. Relationships between strings may also be derived (e.g. upcase). In some cases, we may wish to store relationships between strings, e.g. 'abbreviate' has_synonym 'shorten' (see NH89, pp. 185-7).

With fact tables, the objects being described are always strings, numbers or described objects with parenthesized reference modes. If the relationship type includes a described object type with no parenthesized reference mode, then for communication purposes one may enter a special symbol to denote the described object. For example, the population of a 1:1 reference type may be portrayed explicitly in a *reference diagram* as shown in Figure 5.38.

The table below the schema is a *reference table*. Here Halpin, Jones and Wang are depicted by stick figures. In the context of a lexical reference scheme one could instead depict described entities by unquoted strings (Halpin etc.); however, this would not work for NREs. Note that quotes are required around the entries in the right-hand column since these are being used here to denote strings.

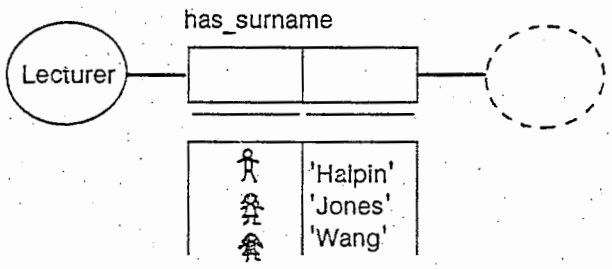


Figure 5.38 A reference diagram

While reference tables are mainly used to help users understand reference schemes, they may also be used to add *simple-existence facts* to the knowledge base for those rare cases involving lazy entities. For example, the reference table of Figure 5.38 translates as the following three simple-existence facts. In the context of the schema, these tell us that there are three lecturers with surnames 'Halpin', 'Jones' and 'Wang'.

∃x(x has_surname 'Halpin')
∃x(x has_surname 'Jones')
∃x(x has_surname 'Wang')

Note that we do *not* count simple-existence facts as elementary facts. Although we have seen that elementary facts are translated as existentially

quantified formulae, in each case the scope of the existential quantifier must be a conjunction of at least two conjuncts. This is true even of unary elementary facts, e.g. ∃x(x has_surname 'Halpin' & Tired x).

So each state of the database may be formulated as a set of facts. In most cases these facts will all be elementary. In those rare cases where lazy entities are allowed, we may have some simple-existence facts. These are the only two varieties of fact that we permit in the database. Our inclusion of simple-existence facts adds to the expressibility of NIAM. Simple-existence facts also simplify the treatment of certain schema transformations, and are easily mapped down to relational database tables.

This completes our basic formalization of, and revisions to, NIAM knowledge base theory. In the next chapter we use this framework to define conceptual schema modalities (e.g. implication and equivalence) and formally prove high level theorems concerning these modalities. These results are later applied with a view to improving the design of relational database schemas.

6 Conceptual schema modalities

6.1 Satisfiability of conceptual schemas

In this chapter we define various modal properties of, and relationships between, conceptual (sub)schemas. This first section focusses on *satisfiability*, while later sections examine constraint implication and subschema equivalence.

A conceptual schema (CS) is a set of KL sentences conforming to the CS formation rules. All permitted graphical components of a CS were discussed earlier, but we have not yet provided formation rules which define which combinations of these components are permitted. A basic set of *NIAM formation rules for a graphical CS* were specified by Leung (1988, sec. 5.3.1). We summarize and refine these in our terminology as follows: each object type plays a role; a pair object type cannot play a role used in its definition; intra-predicate UCs on *n*-ary predicates must span at least *n*-1 roles (we downgrade this from a formation rule to a warning that the fact type is compound if this rule is broken); inter-predicate UCs have a common join object type; subtype graphs are acyclic and no transitively implied links are shown; and subset, equality and exclusion constraints connect compatible (same primitive supertype) sequences of object types.

We have reformulated the last of these rules to allow cases which were excluded in Leung's specification (e.g. an exclusion constraint between the roles of a homogeneous binary -- Leung demanded that the operand predicates differ). We have also omitted the requirement that the CS graph be connected. Typically, connectivity will apply (especially if we allow implicit connection via Real or String); however in some cases it is useful to consider schemas without this property (e.g. arbitrary subschemas).

Some other formation rules may be obtained by verbalizing those aspects of Leung's metaschema (1988, p. 7-5) that are consistent with our formalism. For example, each role is played by exactly one object type, and each predicate has at least one intra-predicate UC. Additional formation rules are needed to cater for our extensions to NIAM, but these are straightforward and are not documented here. However we do consider whether further formation rules should be added to prevent the user from specifying conceptual schemas that are only trivially satisfiable.

As discussed in chapter 3, a CS specifies a structure of interest known as the UoD, which may be regarded extensionally as a set of possible subworlds. Each such subworld provides a dynamic interpretation of the CS. Each state of knowledge (KB state) about one of these subworlds provides a static interpretation of the CS. In this thesis we examine only static interpretations. For convenience we use the terms "interpretation", "equivalence" etc. without qualification to mean static interpretation, static equivalence etc. Our formalization uses the standard notion of an interpretation of a first order theory, restricted to KL sentences.

An *interpretation* of a CS comprises a non-empty domain D of objects (individuals), together with the following assignments: each IC maps to one object in D; each predicate symbol maps to a relation over D; each function symbol maps to a function with arguments and values in D; the operators ~, &, ∨, → and ≡ are given their usual truth-functional interpretations; the quantifiers ∀, ∃ are interpreted by conjunctive and disjunctive expansion over D; = is interpreted as the identity relation; and all other KS constants, predicates, functions, operators, and abbreviations are interpreted as specified for KS earlier.

An interpretation I of a conceptual schema CS, where CS is formulated as a set of KL sentences, is a *model* of CS iff each sentence of CS is true for I. A conceptual schema is *satisfiable* iff it has a model. In practice this notion of satisfiability is too weak, since it permits schemas with constraint patterns that are satisfiable only because these patterns are not populated. For example, consider the KB diagram of Figure 6.1. Although the CS is well formed according to our present formation rules, it is "silly" to assert both the frequency and uniqueness constraint on Teaches (these constraints are labelled C1 and C2 for ease of reference).

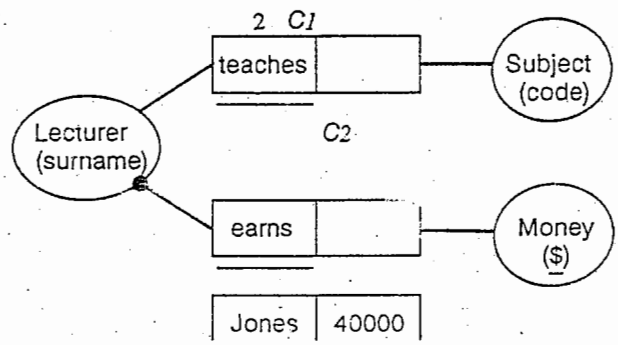


Figure 6.1 Undesirable schemas may be trivially satisfiable

The KB diagram can be used to construct a model for the CS, in which Jones is the only lecturer, earns $40000, but doesn't teach any subjects. So the schema is satisfiable. But asserting constraints C1 and C2 means that anyone who teaches teaches exactly two subjects, and anyone who teaches teaches exactly one subject. This "silly" constraint combination is trivially satisfied in our model, where nobody teaches.

The unsatisfactory nature of trivial models in relation to constraints has been noted in the literature. For example, Meyer, Weigand and Wieranga (1988, p. 13) attempt to avoid the problem by demanding that all models be non-empty. However, this is not a strong enough requirement. In our system, all models are non-empty anyway; but this does not avoid problematic cases like that of Figure 6.1.

It is clearly undesirable to have a knowledge base design which may work with trivial populations only to fail with comprehensive populations. We address this issue by proposing a stronger notion of satisfiability. In preparation, we first define the term "underhired". To keep this discussion brief we make use of familiar set theoretic terminology, and postpone a detailed treatment of relevant cardinality and constraint implication results till the next section.

A set of roles $r_1,..,r_n$ is said to be *underhired* iff the roles are mutually exclusive, their object types are compatible, and their cardinality limit (i.e. the maximum cardinality allowed for the union of the role populations) is less than n. Basically this means we have not "hired" enough actors for all the roles to be played simultaneously. Figure 6.2 illustrates the situation where each role is played by the same object type. The cardinality of the population of this object type sets an upper bound for the cardinality of the union of the n role populations.

The general definition of "underhired" allows that some of the n roles may be played by different subtypes of the same head supertype, and that the mutual exclusion and cardinality constraints might be implied rather than specified explicitly.

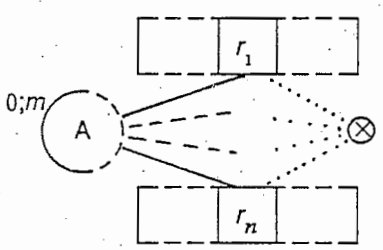


Figure 6.2 If $m < n$ then $\{r_1,..,r_n\}$ is underhired

As an example including both the simple and more complex cases, consider the UoD specified briefly as: President has Gender {'m','f'}; President is_given CompanyCar or is_assigned Taxicard, but not both; MalePresident born_in Year; FemalePresident has Maiden_name; there is only one President. Here it is impossible to simultaneously populate more than one of the predicates is_given and is_assigned, or more than one of the predicates born_in and has_maiden_name. So here we have two underhired role sets.

We now discuss some proposals for a stronger notion of schema satisfiability. To begin with, one might argue that a CS should have a model in which *all* its predicates are instantiated. However, this requirement is too strong, since as we have just seen, legitimate schemas may violate this condition by including underhired role sets. Instead, we demand that each predicate be individually instantiable:

> A conceptual schema CS is *strongly satisfiable* if and only if, for *each* of its predicates, there is a model of CS in which *that* predicate is instantiated.

While strong satisfiability can be proved simply by creatively producing models, a formal proof is required to show that a CS is *not* strongly satisfiable. The CS is formalized in KL and existential clauses are added to produce an arbitrary candidate model for that case; a deduction tree is then used to determine whether the case generates a contradiction; if any of these arbitrary cases generates a contradiction, a model of this type is not possible and hence the CS is not strongly satisfiable. The logician can usually use his/her intuitions to quickly select a candidate case that closes (and save work by translating only the relevant part of the CS which generates the closure). Since such proofs are generally straightforward but often lengthy, we do not include them in the body of this thesis. Appendix II contains some sample proofs of various results: proof 1 is a simple deduction to show that the schema of Figure 6.1 is not strongly satisfiable.

Often the designer can easily "see" that a constraint pattern is not strongly satisfiable, and has no need to check this intuition with a formal proof. The C_1, C_2 pattern in Figure 6.1 is such a case. However, the expressiveness and constraint inter-dependency aspects of NIAM lead to many possible constraint patterns for which it is not intuitively obvious whether the pattern is strongly satisfiable. Indeed, some of my students have even had trouble picking the faults with simple cases like the subschema of Figure 6.3. The subtype definition is omitted as it is irrelevant to our discussion.

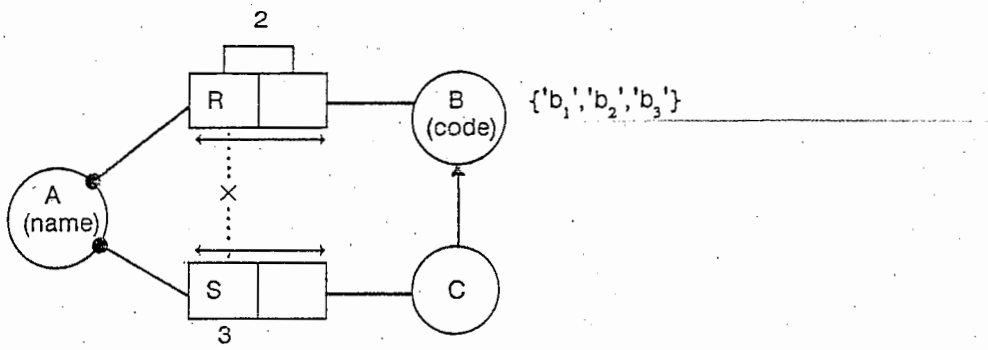


Figure 6.3 This is not strongly satisfiable on three counts

Figure 6.3 has three faults, any of which prevents strong satisfiability. On predicate R, the frequency constraint clashes with the uniqueness constraint (cf. Figure 6.1), and moreover any FC greater than 1 should be rejected if it spans the whole predicate. Secondly, the mandatory roles on A imply an equality constraint between these roles; so if A is populated the exclusion constraint cannot be satisfied (implication is discussed in detail in the next section).

Finally, the FC of 3 on S.1 cannot be satisfied once A is populated. If it were satisfied then C would include at least 3 instances (since each tuple in S is unique); but this is impossible since C is a proper subtype of B (only proper subtypes are allowed in NIAM), and hence C has a maximum cardinality below 3 (because of the lexical constraint on B: we formalize such cardinality-based reasoning in section 6.2).

The argumentation in the two previous paragraphs would typically be accepted as a "high level proof" of the faults in the subschema. In most cases we argue at this level. In cases where some aspects are difficult to see at the high level, or where we feel a need to check our high level reasoning, low level formal proofs by deduction trees or similar mechanisms are used. One aim of this thesis is to lay the formal groundwork for the construction of various interactive schema design modules, including a proof editing environment to support high level reasoning about conceptual schemas.

Once a proof has been constructed to show that a constraint pattern is not strongly satisfiable, it may be desirable to incorporate this result in the set of CS formation rules. Although this adds to the complexity of the CS parser, as well as lengthening the list of formation rules with which the designer should be familiar, in most cases it is better to have such faults detected as early as possible. Moreover, by pushing such results down to the syntax level, the later semantic work is simplified since it can assume the CS is free of these faults.

With this in view, we propose that the following rules be added to the set of graphic CS formation rules. These cover the cases discussed here, as well as some others. We use the term "sequence" and "product" to allow unit cases: a sequence or product of one term is identical to that term. A sequence of *n* roles is "exactly spanned" by a UC iff the UC spans it and no other roles. Intuitively, a role population is the set (not bag) of objects playing that role in the given interpretation. The maximum cardinality of a role population equals the maximum cardinality of the population of its attached object type. Further background justifying Rule 7 is provided in the next section.

Additional CS formation rules:

1 An FC of 1 is never used (the UC form must be used instead).
2 An FC cannot span a whole predicate.
3 No role sequence exactly spanned by a UC can have an FC.
4 No UC can be spanned by a longer UC.
5 An exclusion constraint cannot be specified between roles if at least one of these roles is marked as mandatory.
6 An exclusion constraint cannot be specified between two roles attached to object types one of which is specified as a subtype of the other.
7 An FC with upper bound *n* cannot be specified on a role sequence if *n* is less than the product of the maximum cardinalities of the other role populations for the predicate.

These additional rules are not exhaustive, but we do not discuss other examples here. We leave it as a topic for further research to identify further cases of constraint patterns which are only trivially satisfiable, and compile a more comprehensive list of CS formation rules.

It should be noted that NIAM advises the designer to check each fact type against a sample population. If correctly carried out, this practice helps to minimize the occurrence of schemas which are only trivially satisfiable, since the first condition for strong satisfiability is then satisfied. However, this practice does not eliminate the problem entirely. It is possible, albeit unlikely, for the designer to make a mistake by failing to see that the sample population contradicts some of the constraints. Moreover, the second or third conditions for strong satisfiability might not be satisfied. Finally, it is possible that the designer might make a typographical error when entering the schema into the system.

For such reasons, it is important to make the set of CS formation rules as comprehensive as possible, and to construct an automated CS parser to enforce these rules. If the parser detects a syntax error, it should highlight the violation for the designer, giving specific details, and prompt the designer to edit the design.

There are many cases in which constraints which are not explicitly specified (e.g. by being marked on the CS diagram) are nevertheless implied. In some cases, constraint patterns may be misleading to humans (e.g. because the implied constraints might not be obvious, especially with implied mandatory roles or implied uniqueness constraints). One might address this issue by specifying additional formation rules to reject such cases.

For example, the additional CS formation rules 5 and 6 might be amended to include subset and equality constraints. However, we feel it is more appropriate to have the automated design tool suggest a generally preferred, replacement constraint pattern to the designer, but allow the designer to decide whether to accept the replacement. Implied constraints and preferred constraint patterns are discussed in the next section.

In formalizing NIAM, we distinguished those features that can only be considered globally, so that the theory may be applied to any subschema of interest. Our work on satisfiability, implication and equivalence also applies equally well to subschemas and global schemas. In a practical application where the global schema is large (e.g. a few thousand fact types), the designer typically works with only a small subschema at a time.

Currently, NIAM provides little in the way of "top-down" guidelines for modularizing a global schema into subschemas. Clearly, some of the modularization strategies used in other methodologies could be adopted, if only to provide the designer with a range of manageably sized views of parts of the global schema. For example, modules might be selected according to their function, and "sparse schemas" can be displayed which show object types without their "attributes". For further discussion of conceptual schema abstraction and modularization guidelines with specific reference to the binary-relationship version of NIAM, see Vermeir (1983) and Shoval (1985).

NIAM does facilitate the "bottom-up" approach, allowing the designer to start with any subschema of immediate interest. However, even if each subschema is "validated" it is possible that the global schema might be faulty, since subschemas may impact on one another. We leave the problem of specifying an adequate means of dealing with subschema integration in NIAM as a topic for further research.

Also worthy of further consideration is the notion of *"finite satisfiability"* proposed by Bry and Manthey (1986). They argue that database states should have finite models, i.e. "the constraints have to admit a finite set of (stored as well as derivable) facts". While we feel that at the conceptual level this requirement is too strong (e.g. in our formalization both Real and String are transfinite sets), in practice the restriction does make sense for all stored facts as well as facts (stored or derived) about most Described objects.

Taking an example from Bry and Manthey, suppose the homogeneous binary predicate *works_for* is irreflexive and transitive, and its first role is mandatory. The only models for this relation require denumerably many workers, which is unrealistic. So we ought to complement our notion of strong satisfiability with at least an implementation warning to promote finite satisfiability for described objects. As noted by Bry and Manthey, finite satisfiability is semi-decidable, and semantic tableaux offer one promising approach for automating checking of this property.

6.2 Constraint implication

In this section we define the notion of *constraint implication* with respect to a conceptual schema, identify several important cases of constraint implication, and provide notations for marking implied constraints on schema diagrams. Ideally, automated support should be provided so that the display of implied constraints can be toggled on or off by the user.

We have specified how to translate a NIAM conceptual schema as a conjunction of KL sentences. Let *CS* be any well formed NIAM conceptual schema, and *C* be any well formed, static constraint on *CS* (but not necessarily included as a conjunct of *CS*), where both *CS* and *C* are expressed in KL. Then *CS implies C* iff *C* is true in all models of *CS*: we write this as $CS \Rightarrow C$. If in this context we treat a knowledge base state (UoD subworld glimpse) as a possible world, then "⇒" may be construed as a modal operator for necessary implication, i.e. $\alpha \Rightarrow ß =_{df} \Box(\alpha \rightarrow ß)$.

The detection of implied constraints is important for both conceptual and implementation reasons. For example, the removal of an implied constraint from a CS usually provides a simpler picture for the human designer, and

improves the efficiency of the implementation by avoiding the overhead associated with coding and enforcing the constraint. Additionally, knowing that some constraints are implied by others can assist the designer to reformulate the schema into another one that is equivalent but preferable. For such reasons, considerable research has been conducted on the general problem of detecting implied constraints.

Most of the research in this area has focussed on functional and multi-valued dependencies within the context of relational database schemas as standardly portrayed in terms of a universe of attributes and a collection of data dependencies (e.g. Beeri 1980; Beeri & Kifer 1986). Other kinds of dependency within this relational model have also been studied (e.g. Sadri 1987), but as more kinds of dependency are considered the complexity of the problem rapidly escalates. NIAM is very rich in the range of dependencies it incorporates, and a comprehensive study of constraint implication within NIAM is beyond the scope of this thesis. Such a study would benefit by including a mapping of the major results of the relational research into NIAM. The mapping of the relational model itself into NIAM is fairly straightforward.

NIAM's graphical notation helps the designer to visualize whether a constraint is implied. To show a constraint is not implied, a counterexample is needed (i.e. a case in which the CS is satisfied but the constraint is not). To show a constraint is implied, a formal proof is strictly required, if only to confirm one's intuitions. We now list a number of constraint implication theorems for NIAM: their names start with the letter "I" (for "Implied"). The other non-digit characters in the name suggest the kind of constraint that is implied (A = Asymmetry, E = Exhaustive subtypes, F = Frequency, FD = Functional Dependency, I = Irreflexive, M = Mandatory role, X = eXclusion, S = Subset, U = Uniqueness, # = cardinality). Examples for many of these were discussed in NH89. Their proofs are straightforward; as a trivial example to demonstrate the basic technique, IS1 is proved in Appendix II (proof 2).

IS1 If an object type A plays roles r_1 and r_2, and r_1 is mandatory then a subset constraint from r_2 to r_1 is implied (see NH89, p. 177).

Corollary 1: If roles r_1 and r_2 are mandatory for A, an equality constraint is implied between them (NH89, p. 173).

Corollary 2: If A is an object type playing role r_1 which is mandatory, B is a subtype of A, and B plays role r_2 then a subset constraint from r_2 to r_1 is implied.

IS2 A subset constraint from the role sequence $r_1..r_n$ to the role sequence $s_1..s_n$ implies subset constraints from r_i to s_i, $i = 1$ to n (NH89, p. 181).

Corollary: An equality constraint between the role sequences $r_1..r_n$ and $s_1..s_n$ implies equality constraints between r_i and s_i, $i = 1$ to n.

IX1 An exclusion constraint between role r of predicate R and role s of predicate S implies an exclusion constraint between any role sequence of R which contains r and any role sequence of S which contains s provided the role sequences are compatible (i.e. their corresponding object types are of the same corresponding primitive types) and r and s occupy the same position in these sequences (NH89, p. 181 gives a simple example).

The converses of IS1-2 and IX1 do not hold (e.g. subset constraints from each role of one role sequence to corresponding roles in another sequence do not imply a subset constraint from the whole of the first sequence to the whole of the second). As an aid to visualization, IS1, the binary form of IS2, and one of the binary forms of IX1 are illustrated in Figure 6.4: here the asterisked constraints are implied.

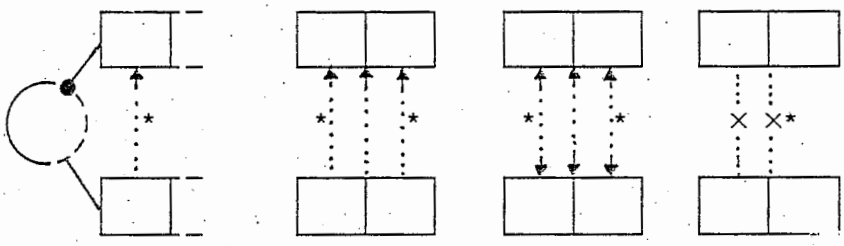


Figure 6.4 Constraints marked '*' are implied

We now consider some implied mandatory roles. IM1 involves not just implied but equivalent representations. In its simplest form there are two roles involved, as shown in Figure 6.5. Consider the left diagram: the subset constraint tells us that any object playing r_2 also plays r_1; the mandatory role disjunction ensures that each object instantiating A plays r_1 or r_2; hence each object instantiating A plays r_1; so r_1 is mandatory, as shown on the right diagram.

IM1 If a disjunction of roles $r_1,..,r_n$ is mandatory and a subset constraint runs from each of $r_2,..,r_n$ to r_1 then r_1 is mandatory. In this case r_1 should be marked as mandatory and these subset constraints should be omitted (NH89, p. 251).

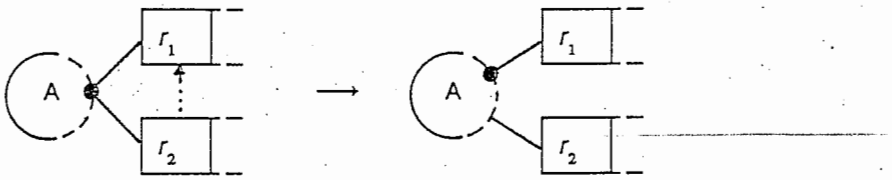


Figure 6.5 An example of IM1: choose the right-hand version

Although equivalence between (sub)schemas is not precisely defined until the next section, it should be fairly clear that both the diagrams in Figure 6.5 express the same information. As indicated in IM1, we suggest that the right-hand version always be chosen: this provides a simpler conceptual picture for the human designer. If desired, an implied MRC may be depicted by marking a *white dot* (i.e. an "o") on the relevant role arc.

Often, more than one constraint implication theorem can be applied. Consider Figure 6.6. Taking the left-hand diagram, IS2 is used to deduce subset constraints from r_1 to s_1 and from r_2 to s_2; then IM1 is applied twice to obtain the right-hand diagram, which is preferred.

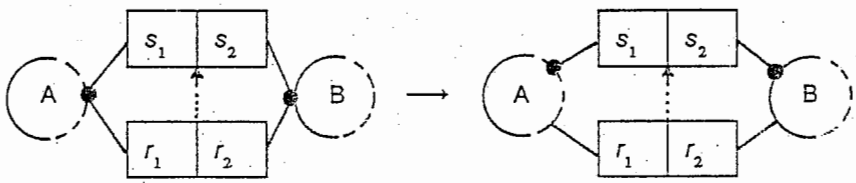


Figure 6.6 Using IS2 and IM1, the right-hand diagram is chosen

IM1 argues for the same display preference in all cases. With most equivalence situations there is a usually preferred choice which may be rejected, for good reasons, in rare cases. Indeed, the right-hand version may itself be transformed by objectification (see later) or by use of a conditional derivation rule (cf. Figure 5.34).

IM2 lists two cases where the preferred version should always be chosen. This theorem is illustrated in Figure 6.7.

IM2 If role r_1 is marked optional, and role r_2 is marked mandatory, and a subset or equality constraint runs from r_2 to r_1 then r_1 is mandatory. In this case r_1 should be marked as mandatory, and the subset or equality constraint should be omitted.

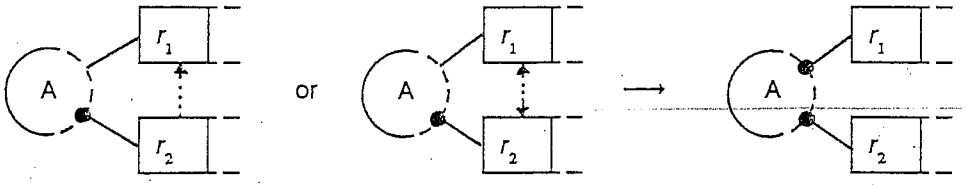


Figure 6.7 IM2: choose the right-hand version

Two applications of theorems IS2 and IM2 are shown for the schema fragments in Figure 6.8. The mandatory role constraints indicated by the white dots are implied by the other constraints. In such cases all the constraints should be shown (the pairwise subset constraint is not implied by the two mandatory roles). Note that the "colour" of the dots matters here: swapping black for white fails, i.e. if the top role played by A is mandatory, and the subset constraint is as shown, it does not follow that the other role played by A is mandatory. Note also that similar results apply if the subset constraint is replaced by an equality constraint.

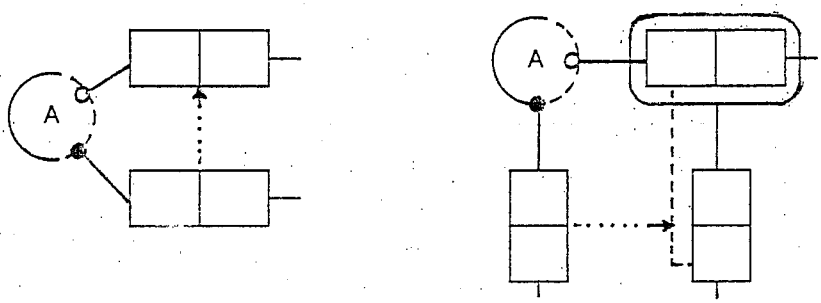


Figure 6.8 The white mandatory role dots are implied

The next three theorems indicate exclusion is stronger than asymmetry which in turn is stronger than irreflexivity. The binary case for each is summarized in Figure 6.9. Proofs of IA1 and II1 for this case are given in Appendix II (Proofs 3, 4). II2 follows directly from IA1 and II1.

IA1 An exclusion constraint between roles r_1 and r_2 implies an asymmetry constraint between them.

II1 An asymmetry constraint between roles r_1 and r_2 implies an irreflexivity constraint between them (NH89, p. 189).

II2 An exclusion constraint between roles r_1 and r_2 implies an irreflexivity constraint between them (NH89, p. 189).

Figure 6.9 The asterisked constraints are implied

In formalizing NIAM we introduced set-like notations as abbreviations without actually positing sets as objects. We now introduce some more abbreviations to help specify later results. The first notation is used for discussing cases where A is a subtype of B, where B plays R.1. English readings for the other notations are shown in parentheses.

A = B R-ing C	$=_{df}$	$\forall xy(xRy \rightarrow Bx) \ \& \ \forall x[Ax \equiv \exists y(xRy \ \& \ Cy)]$	
$x \in A \cup B$	$=_{df}$	$Ax \vee Bx$	{ x belongs to A union B }
$x \in A \cap B$	$=_{df}$	$Ax \ \& \ Bx$	{ x belongs to A intersect B }
$A \cap B = \{\}$	$=_{df}$	$\sim\exists x(Ax \ \& \ Bx)$	{ A and B are disjoint }

In section 2.1 we discussed a simple example of deducing exclusion and exhaustion constraints among subtypes, using the subtype definitions and constraints on the defining fact types. Before specifying the relevant theorems (IX2-3) to cover such examples, we state a simple lemma using standard mathematical language.

Lemma 1: If *f* is a function which maps set A_1 into B_1 and set A_2 into B_2, and B_1 and B_2 are disjoint, then A_1 and A_2 are also disjoint.

To prove this lemma, assume it is false. Then there must be an object, *a* say, in $A_1 \cap A_2$. Since $a \in A_1$, $f(a) \in B_1$. Since $a \in A_2$, $f(a) \in B_2$. So $f(a) \in B_1 \cap B_2$. But this is impossible since B_1 and B_2 are disjoint. So the lemma is true. This lemma is the basis for IX2:

IX2 If R is a functional (*n*:1) relation from A to B, and A_1 and A_2 are subtypes of A defined as R-ing disjoint subtypes of B, then an exclusion constraint is implied between A_1 and A_2 (and hence between any role played by A_1 and any role played by A_2).

This is depicted in Figure 6.10, where the implied exclusion constraint is asterisked and is taken to mean: $\sim\exists x(A_1x \ \& \ A_2x)$. Since we specify theorems in terms of subschemas rather than global schemas, it is not necessary that R.1 be mandatory for A.

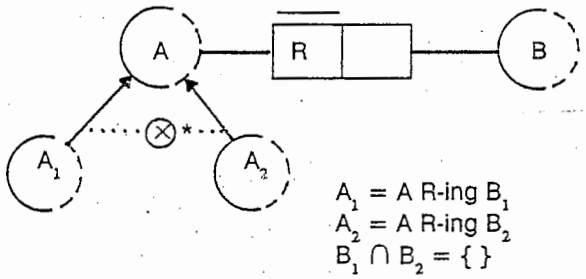


Figure 6.10 The populations of A_1 and A_2 must be mutually exclusive

To apply this theorem in a particular case, we first need to show that B_1 and B_2 (as captured by the explicit definitions for A_1 and A_2) are disjoint. If B is lexical or numeric then the disjointness of B_1 and B_2 can be determined from our lexical and numeric axioms. If B is described, then the disjointness of B_1 and B_2 can be determined by applying Lemma 1 to the reference function which maps B into the lexical/numeric subtypes used to reference B and noting whether these latter subtypes are disjoint.

For example, consider the functional (*n*:1) fact type: Student (name) scores Rating (nr) [1..7]. Define two subtypes: Passer = Student scoring Rating ≥ 3; Failer = Student scoring Rating < 3 (i.e. ∀x[Passer x ≡ ∃yz(x scores y & y has_rating_nr z & z ≥ 3)]; ∀x[Failer x ≡ ∃yz(x scores y & y has_rating_nr z & z < 3)]). The numeric subtypes [3..7] and [1..2] are disjoint. By lemma 1, the Rating subtypes which map via the has_rating_nr function to [3..7] and [1..2] are disjoint. So by IX2, the populations of Passer and Failer, which functionally map (via Scores) to these disjoint Rating subtypes, are mutually exclusive.

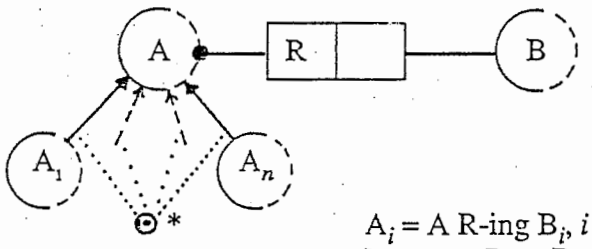
While on the subject of exclusion constraints, we note the following theorem. This is trivially proved from TXC6 and TXC2.

IX3 A mutual exclusion constraint between *n* roles implies an exclusion constraint between any two of these roles.

In general, sets $A_1,..,A_n$ are said to *exhaust* B iff $A_1 \cup .. \cup A_n = B$. We now specify the main situation where the populations of a number of subtypes must (collectively) exhaust that of their common supertype.

IE1 If R is a relation from A to B, R.1 is a mandatory role for A, and $A_1,..,A_n$ are subtypes of A defined as R-ing subtypes $B_1,..,B_n$ which exhaust B, then $A_1,..,A_n$ exhaust A.

This is depicted in Figure 6.11, where the circled dot means $A_1,..,A_n$ exhaust A. Note that while R.1 must be mandatory, it is not necessary that R be functional.



A_i = A R-ing B_i, i = 1 to n
$B_1 \cup .. \cup B_n = B$

Figure 6.11 The populations of $A_1,..,A_n$ must exhaust that of A

A sketch of the proof for Theorem IE1 follows. Assume the conditions are satisfied but $A_1,..,A_n$ are not exhaustive. Then there is some element of A, *a* say, which does not belong to any of $A_1,..,A_n$. Since R.1 is mandatory, *a* maps to at least one element, including *b* say, of B. Since $B_1,..,B_n$ exhaust B, *b* belongs to one of these, say B_i. But A_i is by definition the set of all elements of A which map to B_i. Hence $a \in A_i$, which contradicts our first deduction. Hence the original assumption is false and the theorem is proved.

In applying IE1, if B is lexical/numeric then exhaustion by $B_1,..,B_n$ is simply determined. If B is described, then the 1:1 reference mapping is used to deduce exhaustion from that of the lexical/numeric images. As a simple example, IE1 may be used to prove that Passer and Failer, as defined earlier, exhaust Student. Note that for this example, both IX2 and IE1 apply; so Passer and Failer form a *partition* of Student.

We now consider *implied cardinality constraints*, and related issues concerning cardinality. The cardinality of a set is the number of distinct elements in it. In Appendix III we use the notation "n(*A*)" for the cardinality of a set *A*. However, in the body of the thesis we use the Z notation for this function, i.e. "#*A*". Syntactically, we treat # as an operator with minimum scope, so brackets are needed if # is applied to complex set expressions. For example: #{5,3} = 2; #({5,3} ∪ {5,7}) = 3.

We determine the cardinality of a *lexical or numeric* object type as follows. If $o_1,..,o_n$ are all string constants or all numeric constants, then $\#\{o_1,..,o_n\} = n$. For closed integer subranges, #[*n*..*m*] = *m*-*n*+1, e.g. #[3..7] = 5. Though rarely used, cardinalities are easily obtained for the other lexical and digital subtype notations (sec. 4.3). For example: each of #<*n*d> and #<d*n*> equals 10^n; if there are *k* letters then #<*n*a> = k^n, and #<a*n*> = *k* +

$k^2 + .. + k^n$; and $\#[\pm d n.d m] = 2 \times 10^{n+m} - 1$. In principle, though not in practice, transfinite cardinalities ($\aleph_0$ and $\aleph$ respectively) are assigned to open integer ranges (e.g. [*n*..]) and continuous subranges of Real.

For a given KB state, an object type *A* is (statically) interpreted as its *population* (i.e. the set of objects instantiating *A* in that state). Though for brevity we often speak of imposing constraints on "types", we understand this to mean that, for each KB state, the constraints apply to populations of the types. Recall (sec 4.5) that a cardinality constraint of 0;*n* on *A* means $\exists^{0;n}x\ Ax$ (i.e. for any given KB state, there are at most *n* objects in the population of *A* for that state). Similarly, #*A* is the *cardinality of the population of A* (i.e. the number of objects instantiating *A* in that KB state). So asserting that $\#A \leq n$ is equivalent to imposing a cardinality constraint of 0;*n* on (the population of) *A*.

If we ever wish to talk non-statically about an object type *A* itself (i.e. the set of all possible objects of that type, including all possible states) we may denote this by "type*A*". From the KS axioms, and our agreement that terms always refer, it follows that the object types String and Real and their subtypes are fully populated in all KB states: their extension and hence interpretation is fixed. For any lexical or numeric type *A*, #*A* always equals #type*A*.

However, a *described object type A* (e.g. Lecturer) may have different extensions in different KB states: the population of *A* may vary from state to state, and hence so may #*A*. In rare cases, cardinality constraints might be explicitly specified for *A*. Typically however, upper bounds for #*A* have to be deduced. There are many ways in which such implied cardinality constraints may be deduced. The most general method maps the population of the described object type into a sequence of lexical or numeric types via its injective primary reference scheme and then uses the product of the known cardinalities of these types as an upper bound. We now specify this method in more detail.

Each described object type has an injective (1:1 into) primary reference scheme in which it plays *n* roles, where $n \geq 1$, and each nested object type formed from an *n*-ary predicate is referenced via a 1:1 function whose argument types are the object types playing the roles of the defining predicate. So any described object type *A* has an n-part primary reference scheme providing a direct injective map to a sequence of object types $T_1,..,T_n$, say, where the T_i are not necessarily distinct. If any of these T_i is described, map it likewise; continue until *A* is mapped into a sequence of string/numeric types. Any population of *A* must inject to the set of tuples of this type sequence.

We define a *head type* to be any object type which is not specified as a subtype. If *A* is a described head type then, in the absence of stronger constraints on its cardinality, any tuple from the type sequence $T_1,..,T_n$ might be used in referencing a member of *A*. So, if *A* is a described head type, the cardinality of the set of all such tuples determines *dmax#(A)*, the *default maximum cardinality* for the *population* of *A*. The above reasoning is summarized by the following pseudocode specification for the recursive function dmax#, and by theorem IS11.

```
Function dmax#(A: described_head_type): Posint;
  { return default maximum cardinality for population of A }

  var cardinality: Posint; { evolves into the required cardinality }

  function image(A: described_object_type): object_type_sequence;
    { return the sequence of lexical and/or numeric object types
      to which A ultimately injectively maps }
    begin
      replace A by the sequence of object types T1,..,Tn
      to which it directly injects via its n-part primary reference scheme;
      for each Ti
        if Ti is described then Ti := image(Ti);
      return
    end;

Begin
  cardinality := 1;
  for each object type T in image(A)
    cardinality := cardinality * #T;              { T is lexical/numeric }
  return cardinality
End.
```

I#1 If *A* is a described head type, $\#A \le$ dmax#*(A)*

For example, from Gender (code) {'m','f'}, Mass (kg) [50..150] and Room (floor#,cell#) <d,d2> we deduce dmax#(Gender) = 2, dmax#(Mass) = 101 and dmax#(Room) = 1000. Hence #Gender ≤ 2, #Mass ≤ 101 and #Room ≤ 1000. If a stronger cardinality constraint is specified, this overrides the default. For example, if 0;500 is specified for Room, then #Room ≤ 500.

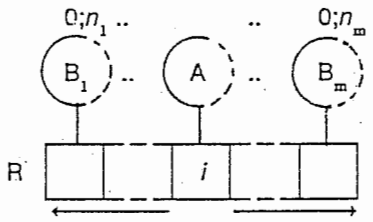
For each KB state, $\#A \le$ #type*A*. If *A* is described, the CS does not include an explicit equation to fix the value of #type*A*. However, if *A* is a described head type, we may equate #type*A* with dmax#*(A)* if we assume that no object in *A* can change its (state-unqualified) primary identifier. This assumption is usually justified, but in rare cases there may be exceptions.

For example, suppose lecturers are identified by their name, and a woman lecturer changes her surname when she marries; or one lecturer leaves and another with the same name arrives. In practice, these cases can be avoided by choosing a state-independent primary reference scheme (e.g. employee#). Even if the primary-reference scheme for *A* is state-dependent (i.e. identifiers must be qualified by the state), dmax#(*A*) is still an upper bound for #*A*; and this is the main issue.

In those rare cases where the environment dictates a change in the primary reference scheme itself (e.g. metrication of the unit system, or amalgamation of two systems with common object types but different reference schemes) special arrangements are needed (e.g. conversion rules). This is one aspect of the general problem of catering for cases where the CS itself evolves. Though this problem is an important one in practice, we do not explore this point further.

The next theorem (I#2) is specified diagrammatically. The cardinality of a role is the number of objects playing that role. In the case shown, R has arity $m+1$ and is played by object types A and $B_1,..,B\mu$. The asterisked role cardinality constraint is clearly implied, since the number of possible permutations of $B_1,..,B_n$ is the product of their cardinalities, and the uniqueness constraint across all but A's role implies that each permutation is associated via R with at most one member of A.

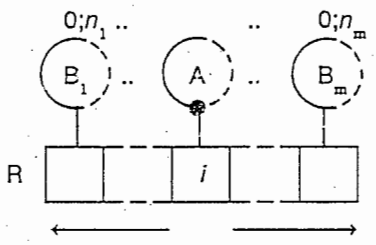
I#2



* $\#R.i \le n_1 \times .. \times n_m$

As a corollary, if R.*i* is mandatory for this situation, then the cardinality of A must be at most the product shown, since all members of A's population play R.*i*. This result is specified graphically as I#2'.

I#2'



* $\#A \le n_1 \times .. \times n_m$

If instead of a UC on the permutations of $B_1,..,B_n$ we have a frequency constraint upper bound u, then each permutation may be associated with u objects from A. Hence:

I#3 If the UC in I#3/I#3' is replaced by an FC with upper bound u, then the product $n_1 \times .. \times n_m$ should be replaced by $u \times n_1 \times .. \times n_m$.

Let us use the term "*max#*"*A* for the *maximum permitted cardinality of the population of A*, for any object type *A*. If *A* is lexical or numeric then max#*A* equals #*A* which equals #type*A*. If *A* is a described head type then max#*A* equals dmax#(*A*) by default; if the cardinality constraint 0;*n* is asserted on *A*, and $n <$ dmax#(*A*) then max#*A* = *n*.

Since NIAM permits only proper subtypes, if *A* is a subtype of *B* then #type*A* < #type*B*, and #*A* ≤ #*B*. Although for a given state it is possible that #*A* equals #*B*, the maximum permitted cardinality of *A*'s population is less than that for *B*. This general result (I#4) is independent of the specific definition for *A*, and was used earlier in discussing Figure 6.3.

I#4 If *A* is a subtype of *B* then max#*A* < max#*B*.

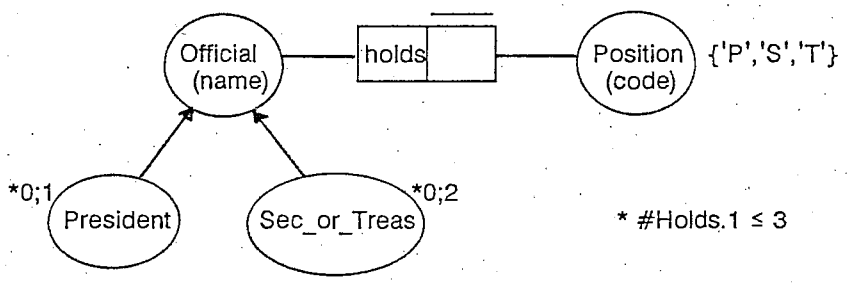
If *A* is a *described subtype*, the definition of *A*, combined with constraints on the predicates participating in this definition, can often be used to deduce an upper bound for #*A* that is smaller than max#*B*-1 where *B* is its smallest supertype. We now state the main results in this regard. These are intuitively obvious, and their formal proofs are straightforward.

I#5 If the *n*-ary predicate *R* is played by object types $B_1,..,B_n$ (not necessarily distinct), a uniqueness constraint of length *n*-1 exactly spans all but the *i*th role, and *A* is defined as the subtype of B_i which plays *R* with specified subtypes $B_1',..,B_{i-1}',B_{i+1}',..B_n'$ of the other types, then $\#A \le \#B_1' \times .. \times \#B_{i-1}' \times \#B_{i+1}' \times .. \times \#B_n'$.

I#5 is obvious since the UC provides a function from the specified population sequence to the population of *A*. If instead of a UC we have an FC with upper bound *u* then each member of the population sequence may be associated with *u* objects in B_i. Hence:

I#6 If the UC in I#5 is replaced by a frequency constraint with upper bound *u*, then $\#A \le u \times \#B_1' \times .. \times \#B_{i-1}' \times \#B_{i+1}' \times .. \times \#B_n'$.

As a simple example of I#5 and I#2, consider Figure 6.12. Here the same official may be president, secretary and treasurer, but each of these positions is held by only one official. So there is at most one president, and at most two officials fill the other positions. These upper bounds on #President and #Sec_or_Treas are shown as implied cardinality constraints, using asterisks. The implied constraint on #Holds.1 follows from I#2.



∀x[President x ≡ ∃y(x holds y & y has_positioncode 'P')]
∀x[Sec_or_Treas x ≡ ∃y(x holds y & y has_positioncode 'S'
∨ y has_positioncode 'T')]

Figure 6.12 The asterisked cardinality constraints are implied

As a simple example of I#6, suppose the UC on Holds is replaced with an FC of 1;2. This allows at most two presidents, at most two secretaries, and at most two treasurers. Clearly, the implied cardinality constraints on the population of President and Sec-Treas are 0;2 and 0;4 respectively.

In sections 4.5 and 5.1, we discussed role-object constraints to specify cases where a UC or FC applies to only some of the objects playing a role (TROC2-3). These constraints can be used to set an upper bound for #1 in the absence of a UC or FC on the other n-1 roles. I#7 sets out the simple case.

I#7 If a subtype *A* consists of those objects playing one role of a binary predicate *R* with an object which has a role-object UC, or FC of upper bound *m*, with respect to the other role of *R*, then $\#A \leq 1$ or $\#A \leq m$ respectively.

Before illustrating I#7, we consider one more result. In general, if A and B are sets then the cardinality of their union is the sum of their cardinalities minus the cardinality of their intersection, i.e. $\#(A \cup B) = \#A + \#B - \#(A \cap B)$. In naming subtypes based on unions/intersections we often include "or"/"and" (e.g. "Sec_or_Treas"). The wording of the following theorem reflects this practice (of course it does not matter what the subtypes are called).

I#8 If *AorB* and *AandB* are equated with $A \cup B$ and $A \cap B$ respectively, then $\#AorB = \#A + \#B - \#AandB$.

As an example of I#7-8, suppose that the Holds predicate of Figure 6.12 is *m:n* (i.e. the UC exactly spans both roles), but that role-object constraints are applied to position codes (and hence to positions) as follows:

$$\{'P'^{1;2}, 'S', 'T'\}$$

Now we have at most one president, at most one treasurer and at most two secretaries. From I#7 the cardinality constraint 0;1 is implied for the subtype President; and if subtypes for Secretary and Treasurer were introduced they would have implied cardinality constraints of 0;2 and 0;1 respectively. For this UoD the same official might be both a secretary and the treasurer; when Secretary and Treasurer are fully populated it is possible that their intersection is empty. If we add a UC on the first role of Holds, we know this intersection is empty. In either case, I#8 yields a cardinality constraint of 0;3 on Sec_or_Treas. If a textual constraint is added to assert that anyone who is treasurer must also be a secretary then I#8 yields a cardinality constraint of 0;2 on Sec_or_Treas.

Suppose however that the role-object constraints are as just stated but an FC of 2 applies to the first role of Holds, and this role is now mandatory (it could have been optional before since Figure 6.12 is a subschema). In this UoD an official holds either no position or two positions. The role-object constraints now imply that the only way of filling all positions is to have exactly two officials: one is the President and a Secretary; the other is Treasurer and a Secretary. But this implies that Sec_or_Treas equals Official, which violates the metarule that only proper subtypes are allowed (cf. I#4). Hence this subschema would not be strongly satisfiable.

As the previous example demonstrates, the full implications of a particular constraint pattern may not be immediately obvious. This can also be the case with *implied uniqueness constraints*. Consider Figure 6.13. Here we have two binaries connected by a binary subset constraint (each ordered pair in R must also be in S). A uniqueness constraint on one of the target roles implies a UC on the corresponding source role. As an alternative to marking the implied UC with an asterisk (which might perhaps be mistaken to mean the fact type itself is implied) we use a *broken uniqueness bar*: its arrow tips may be omitted if the roles are contiguous.

IU1 If a binary subset constraint runs from the binary predicate R to the binary predicate S, then a UC on S.*i* implies a UC on R.*i*, *i* = 1 to 2.

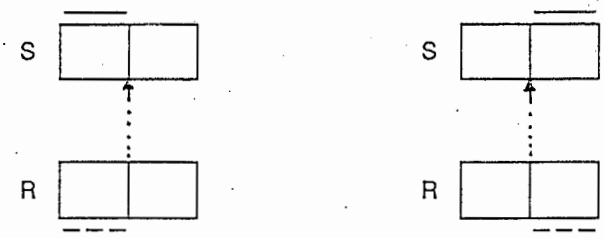


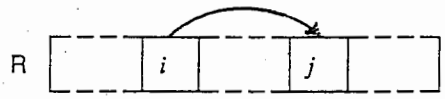
Figure 6.13 The UCs shown as broken bars are implied

A sketch of the proof for the left hand part of IU1 follows. Assume the implied UC does not hold. Then there is a state in which Ra_1b_1 and Ra_1b_2, for some a_1, b_1, b_2. The subset constraint implies that in this state Sa_1b_1 and Sa_1b_2, which contradicts the UC on S.1. So the assumption is false, i.e. there is a UC on R.1. Similarly, the right hand part of the theorem is proved.

Note that for the subset constraint pattern of Figure 6.13, a UC on a source role does *not* imply a UC on its target role. Non-implication may be shown by citing a counterexample. One counterexample here is: Ra_1b_1; Sa_1b_1; Sa_1b_2; Sa_2b_1. In this state, the subset constraint is satisfied as well as UCs on each of the source roles, but neither target role has a UC.

Note also that with unary subset constraints, a UC on one role (source or target) does *not* imply a UC on the role at the other end of the subset constraint. For example, in the following state, subset constraints run from R.1 to S.1 and R.2 to S.2 and UCs apply to S.1 and S.2, but UCs do not apply to R.1 or R.2: Ra_1b_1; Ra_1b_2; Ra_2b_1; Sa_1b_1; Sa_2b_2.

In most modelling approaches, the most important kind of constraint is the *functional dependency* (FD). Within our framework we may speak about an FD from one role(sequence) to another role. The simple case of an intra-predicate FD from one role to another may be specified as shown in Figure 6.14, using a solid arrow to show the direction of the FD. This means each object instantiating R.i is associated via R with only one object in R.j. If A_i and A_j play R.i and R.j respectively, we may express the FD shown by saying that A_i *functionally determines* A_j *in* R.

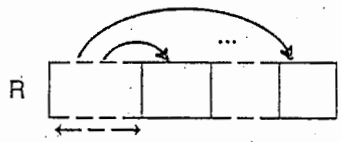


$\forall xy\underline{x}\ [\ R\underline{x}\ \&\ x_i = x\ \&\ x_j = y \rightarrow \forall\underline{x}(R\underline{x}\ \&\ x_i = x \rightarrow x_j = y)\ ]$

Figure 6.14 An FD from role *i* to role *j*

The following theorem is obvious, since a role(sequence) governed by a UC can only be instantiated once.

IFD1 A UC on a role(sequence) in a predicate R implies FDs from this role(sequence) to each other role in R.



The converse of this theorem also holds (IU2). The proof is trivial: if the FDs exists, each instantiation of the source role(sequence) is associated with a single object for each of the other roles; if the source is duplicated then the same target objects obtain, which contradicts the default UC across the whole predicate.

IU2 Given any predicate R, if FDs stem from a role(sequence) in R to each other role in R then a UC spans this role(sequence).

To help ensure that all fact types are elementary, NIAM demands that no fact type has a UC which fails to span two of its roles (see NH89, sec. 5.2). A UC on a single role provides a *simple key*, and a UC spanning more than one role provides a *composite key*. Clearly, only binary fact types can have a simple key.

In NIAM, so long as the fact types actually are elementary, there is no need to introduce a special notation for functional dependencies within a fact type, since if a fact type is elementary all its FDs are captured by its UCs. If we find the need to assert an FD on a fact type which is not captured by a UC then we know the fact type is not elementary, and we should split it on the source of the FD. Background on such "splitting" is given in section 5.2 of NH89.

Suppose we have a situation like that of Figure 6.13 except that R is longer than a binary. By a proof similar to that for IU1 it may be shown that an FD is implied as shown by an asterisked solid arrow in Figure 6.15. But

this entails that R is not elementary. R should thus be split on the object type playing the source role for the FD. The result is specified as theorem IFD2.

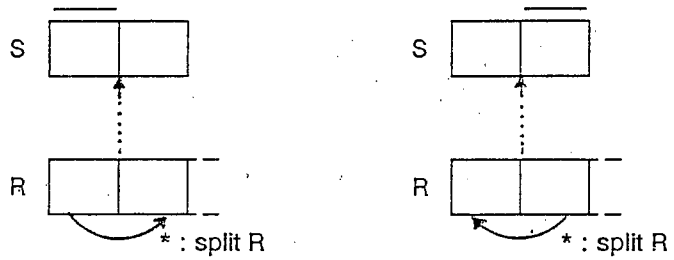


Figure 6.15 Because of the implied FD, R should be split

IFD2 If R is at least a ternary, and a binary subset constraint runs from R.1,2 to the binary predicate S, then a UC on S.1 (or S.2) implies an FD from R.1 to R.2 (or from R.2 to R.1). In this case R should be split on the object type playing R.1 (or R.2).

Clearly, IFD2 may be generalized so that the source roles for the subset constraint may occur in any position of the predicate R. We discuss an important application of IFD2 in the next chapter (a Lecturer-Subject-Student schema requiring Boyce-Codd Normal Form). This application also illustrates the next theorem.

IU3 If role *i* of predicate R is functionally dependent on some other role of R then a UC across all but the *i*th role is implied.

For example, consider Figure 6.16. If the implied UC did not hold then it would be possible to have a state where $Ra_1b_1c_1$ and $Ra_1b_2c_1$, which contradicts the FD from R.1 to R.2.

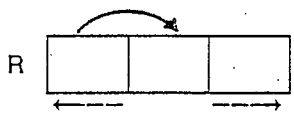
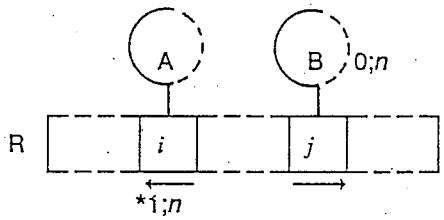


Figure 6.16 The UC is implied by the FD, and R is splittable.

If the predicate R in IU3 is binary, the theorem is equivalent to IFD1. If R has a higher arity then R is splittable on the source of the FD. This theorem can be useful in examining questionable fact types, as well as composite fact types in output reports.

We now briefly consider *implied frequency constraints*. The most useful theorem in this category is as follows. Its proof is trivial.

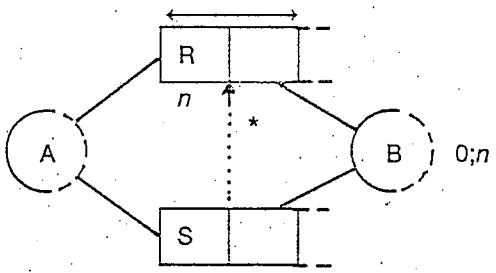
IF1 If roles i and j of R are played by A and B, a UC exactly spans these roles, and #B ≤ n, then role i has an implied FC of 1;n.



Other examples of implied frequency constraints are discussed in NH89 (pp. 152-3). Because of the large variety of constraint categories in NIAM, the constraint implication theorems we have considered in this section are not exhaustive. However, the theorems we have considered do have substantial practical application. We recommend the development of further theorems of this nature as a worthwhile research activity.

Before closing this section we mention two more theorems (IS3 and IU4) that are relevant to specific schema optimization examples discussed in the next chapter.

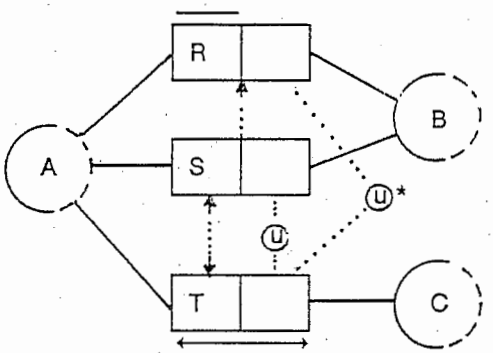
IS3 The subset constraint in the following figure is implied by the other constraints.



The proof of IS3 is simple. Because of the UC, the frequency constraint and the cardinality constraint, any object a playing R.1 plays with all objects in the type B. Since R.1 is mandatory, all (a,b) pairs in the database must occur in R.1-2. Hence any pair in S.1-2 must occur in R.1-2. We make use of this theorem in the Zoo example of section 7.2.

The last theorem (IU4) is fairly specific. We cite it here to shorten discussion of a BCNF example in section 7.2.

IU4 The asterisked UC in the following diagram is implied by the other constraints.



The proof of IU4 is by reductio ad absurdum. Assume the asterisked constraint does not hold. Then without loss of generality, there are 2 tuples (a1,b1,c1) and (a2,b1,c1) ϵ R*S. Hence {(a1,b1),(a2,b1)} $\subseteq$ R and {(a1,c1),(a2,c1)} $\subseteq$ T. Because of the equality constraint, {a1,a2} $\subseteq$ S.1. Because of the subset constraint, and the UC on R, a1 and a2 must be paired with b1 in S (pairing with anything else, say b2, would imply a1Rb2 which violates the UC on R). But this violates the inter-predicate UC between S and T. Hence the inter-predicate constraint between R and T must hold.

6.3 Equivalence of conceptual schemas

In this major section we discuss *equivalence* between conceptual (sub)schemas. Although limited to static interpretations, this notion caters for the possibility that the schemas may differ in their predicates. Our formalization is used to establish several important equivalence theorems. These theorems are applied in the next chapter to perform conceptual optimization. Further background on schema equivalence in NIAM is given in chapter 10 of NH89.

Considerable research has been conducted on the general problem of determining whether two schemas are equivalent. However the bulk of this research has focussed on the equivalence of relational schemas (e.g. see Kobayashi 1986). Some work has also been done on equivalence between conceptual schemas based on ER-modelling (e.g. D'Atri & Sacca 1984). Within NIAM, four basic equivalence results have been stated (e.g. Falkenberg 1986, pp. 7-14/20) but, apart from our own work, we are not aware of any formal treatment of these results. Our formalization enables NIAM equivalence theorems to be precisely specified and rigorously proved. We also introduce some new equivalence theorems. Appendix III briefly considers how our results may be used within the ER framework.

Let T_1 and T_2 be conjunctions of KL sentences, and let "model" abbreviate "static KS model". Then T_1 *implies* T_2 iff each model of T_1 is also a model of T_2, i.e. $T_1 \Rightarrow T_2 =_{df} \Box(T_1 \rightarrow T_2)$. T_1 and T_2 are *equivalent* iff they have exactly the same models, i.e. $T_1 \Leftrightarrow T_2 =_{df} \Box(T_1 \equiv T_2)$. Recall that KS is a first order theory. T_1 and T_2 are equivalent if each may be proved from the other in KS. So "equivalent" is short for "KS-equivalent".

Let CS1 and CS2 be two conceptual schemas (either subschemas or global schemas), each of which is expressed as a conjunction of KL sentences. Let K be the conjunction of the axioms of KS. Let S1 and S2 be the specific axioms of CS1 and CS2 respectively, expressed as a conjunction of KL sentences. So CS1 is K & S1, and CS2 is K & S2. Let R1 and R2 be the sets of predicate and function symbols of CS1 and CS2 respectively, that are not used in K.

Since the KS axioms are included in all schemas, the languages of CS1 and CS2 can differ only in their predicate and function symbols. If R1 equals R2, then equivalence between CS1 and CS2 is defined as previously (by default, symbols common to both are given the same interpretation).

If R1 differs from R2, or some symbol common to R1 and R2 is to be interpreted differently in the two schemas (such schemas cannot be included in the same global schema), then implication and equivalence between CS1 and

CS2 can only be specified within the *context* of rules which translate the predicate and function symbols of each purely in terms of the symbols of the other. If such rules are supplied, then it is useful to know whether the schemas are equivalent in the context of these rules.

Let D1 be a conjunction of KL sentences defining the symbols of R1 purely in terms of the symbols of CS2. Similarly let D2 define R2 purely in terms of the symbols of CS1. Then CS1 is *contextually equivalent* (under D1/D2) to CS2 if and only if CS1 & D1 ⇔ CS2 & D2.

We set out definitions of predicate/function symbols as universally quantified biconditionals/identities. The definiens is always shown as the left operand. Definitions for symbols common to R1 and R2 may be omitted, with the tacit understanding that they have common interpretations.

The theories CS1 & D1 and CS2 & D2 provide a *conservative extension* to the theories CS1 and CS2 respectively. The extension is conservative since no new primitives are introduced. For formal background on conservative extensions and isomorphic embeddings see Keisler (ed. Barwise 1977, p. 56), Chang & Keisler (1977, Ch. 3) and Hunter (1971, pp. 201-5).

Conceptual equivalence between (sub)schemas licences the interchange of their definitionally extended forms, usually for conceptual simplicity or later implementation efficiency. In practice, this usually takes place within a large global schema. It is possible that some predicate or function symbols in the replacing subschema occur in the global schema but not in the replaced subschema (except in the contextual definitions). In this case the interpretation given to these symbols in the contextual definitions must agree with their interpretation in the global schema. If the symbols have different readings in the two contexts then renaming to avoid this ambiguity is required before the replacement.

Before specifying general purpose equivalence theorems we clarify, by means of examples, some finer points concerning object type introduction. To begin with, consider the subschemas in Figure 6.17.

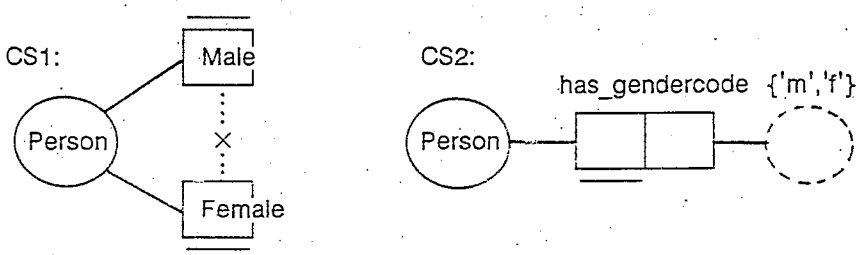


Figure 6.17 Is there a context in which CS1 and CS2 are equivalent?

Here the predicates Male and Female occur in CS1 but not CS2, and the predicate Has_gendercode appears in CS2 but not CS1. The reference mode for Person has been omitted since it is not relevant to our discussion. Leaving aside some finer points, CS1 roughly says that nobody can be both male and female, and CS2 roughly says that a person can have only one gendercode and this must be "m" or "f". Using our intuitive understanding of the connection between these predicates, we might feel that CS1 and CS2 are equivalent since they appear to "say the same thing" in different ways.

While useful, this kind of "gut feeling" approach towards equivalence suffers from two major drawbacks. Firstly, our gut feelings are not always reliable: we might miss some subtle aspect that is captured in one representation but not the other. Secondly, if we wish to have computer support for determining equivalence, we need to formally capture our intuitive connections between the predicates. As Brachman (1988, p. 10) says, "Intuitions about the meanings of English words, which are available to us and allow us to make inferences without conscious effort, are not magically present in the machine". So we need to specify definitions to translate between the predicates used in the different schemas.

An appropriate specification is set out in Figure 6.18. For convenience, each predicate has been abbreviated to its first letter.

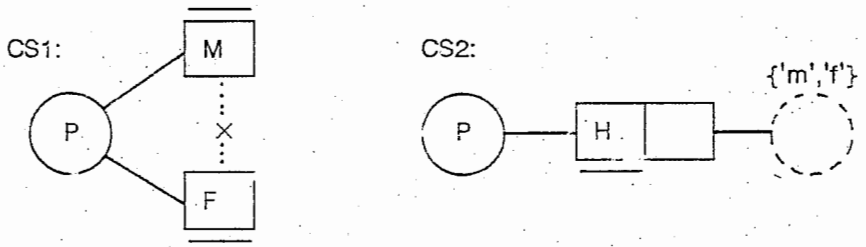


Figure 6.18 CS1 & D1 ⇔ CS2 & D2

Human intuition is required to provide candidate formulae for D1 and D2; however, once stated, they can be formally tested. The formulae in D2 define Male and Female in terms of the language of CS1 ('m' and 'f' are individual constants in both schemas since each schema embeds K). The D2 formulae are easily produced. The definition D1, though obvious in hindsight, is harder to create.

From our experience, we suggest the following method for developing formulae for D1 and D2 for other cases. Use your intuitions to make an

educated guess for D1 and D2. Test this by means of a deduction tree for CS1 & D1 ⇔ CS2 & D2. If a permanently open branch indicates failure, use the counterexample generated from this branch to produce new candidates for D1 and D2 which avoid this counterexample. Continue this refinement-by-counterexample process until the formulae chosen are proved correct. We regard counterexample generation to be one of the main benefits of approaches which incorporate semantic tableaux.

A formal proof for the assertion in Figure 6.18 that CS1 and CS2 are contextually equivalent under D1/D2 is given in Appendix II (proof 5). To perform the proof, the subschemas must first be translated into KL using the rules provided in earlier chapters. This gives:

CS1:	∀x(Px → Described x)	CS2:	∀x(Px → Described x)
	∀x(Mx → Px)		∀xy[xHy → Px & (y='m' ∨ y='f')]
	∀x(Fx → Px)		∀xyz(xHy & xHz → y=z)
	∀x~(Mx & Fx)		
	K		K

Since K is included, we know that 'm' ≠ 'f' (from theorem CC≠). This is needed. The deduction tree proof shown in the Appendix is now straightforward, but lengthy (three pages of formulae). Such proofs can be facilitated by automated support. Substantial research has been carried out to develop automated theorem provers (e.g. see Bledsoe & Loveland eds. 1984) and interactive proof editors (Robinson & Staples 1988), including automated support for semantic tableaux (e.g. see Reeves, 1987). Lindsay (1988) provides a survey of computer support for formal reasoning. Our own experience with proof generation indicates that the development of a proof editor specifically designed to assist reasoning about NIAM conceptual schemas would be of considerable research value. Though not discussed in this thesis, we have begun design and coding efforts in this regard.

Because the definitional rules are bi-directional, both defined and defining predicates can, in principle, be used for both update and query. Behaviourally, CS1 & D1 and CS2 & D2 will generate the same responses for any update/query. In typical practice however, once a choice has been made, only the defining predicate is used for update. To indicate this, the definition may be asterisked, since it then has the implementation status of a derivation rule.

Once a subschema has been replaced by another to which it is contextually equivalent, the designer will often decide that contextually defined

predicates unique to the replaced subschema are no longer of interest. In this case, a deliberate decision may be made to drop the defined predicates from the schema. For example, after transforming from the unary to the binary version in Figure 6.18, the designer might decide to drop the rules in D2. This loss of context means that equivalence is also lost. For example, assuming people are referenced by surname, the following conceptual sentence may be used in an update or query for CS1, but not for CS2 without D2: ∃x(x has_surname 'Halpin' & Male x).

So long as the designer is aware of the context in the first place (as specified in our equivalence theorems), he or she will be aware of precisely what information is being discarded, and hence can maintain conscious and precise control over the semantic impact of schema transformations.

In some cases, contextually equivalent subschemas may both be present within the same global schema. In this case, if D1 (or D2) is a theorem of the global schema, then S2 (or S1) may be deleted, provided its impact (if any) on the global schema is catered for. The need to cater for global impact when making local changes also arises when one subschema is replaced by another, since the original subschema may share constraints with its environment (e.g. inter-predicate constraints and nesting). Let's consider a simple example.

A science fiction novel by Asimov (1986) describes a planet Solaria, whose people are hermaphrodites. Suppose our UoD includes both humans and Solarians, and we use the terms "male", "female" and "hermaphrodite" in a mutually exclusive sense. We might portray this situation by Figure 6.19.

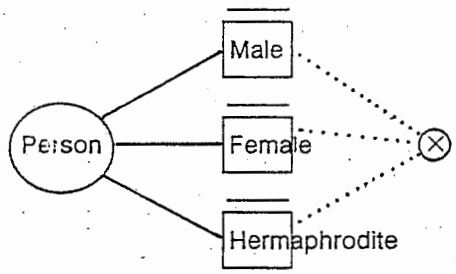
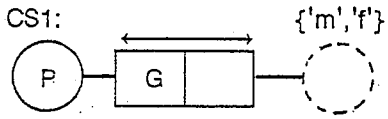


Figure 6.19 A UoD with both humans and Solarians

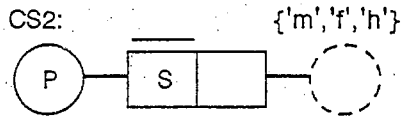
Since an exclusion constraint is implied between Male and Female, the equivalence theorem of Figure 6.18 may be applied to the subschema containing just the top two fact types. However, to preserve the stronger exclusion constraint of Figure 6.19, an exclusion constraint must now be added between the Hermaphrodite predicate and the first role of Has_gendercode.

Of course, a better transformation would be to replace the three unaries with a single binary (Has_sexcode) using {'m','f','h'} (see right half of Figure 6.20). Note that this binary has a simple key: this desirable feature derives from our decision to make the three sexes mutually exclusive, rather than defining a hermaphrodite to be both male and female, which would lead to a composite key (see left half of Figure 6.20). The contextual equivalence between the two versions is as shown. The predicates Has_gendercode and Has_sexcode are abbreviated as "G" and "H". Transforming composite keys into simple keys is useful for reducing the number of tables required in relational databases.

The notation "*CWA" below CS1 means the relevant closed world assumption is required for CS1 (a hermaphrodite cannot have just one sexcode recorded). For some recent discussion of the closed world assumption and incompleteness in the relational model see Motro (1986) and Gottlob & Zicari (1988).



D1:
$\forall xy[xSy \equiv \exists!y(xGy \ \& \ (y='m' \vee y='f')) \vee xG'm' \ \& \ xG'f' \ \& \ y='h']$
*CWA



D2:
$\forall xy[xGy \equiv y='m' \ \& \ (xS'm' \vee xS'h') \vee y='f' \ \& \ (xS'f' \vee xS'h')]$

Figure 6.20 The right-hand version is usually preferred

If a lexical or numeric subtype is given a specific name in just one of the schemas then this information must be included in the contextual definitions below the other schema. For example, suppose {'m','f'} in Figure 6.18 is given the name "G" (abbreviating "Gendercode"). The equivalence CS1 & D1 $\Leftrightarrow$ CS2 & D2 holds so long as we rephrase S2 and D1 to include this new feature, i.e.

S1: $\forall x(Px \rightarrow$ Described x)
$\forall x(Mx \rightarrow Px)$
$\forall x(Fx \rightarrow Px)$
$\forall x \sim(Mx \ \& \ Fx)$
D1: $\forall x(Gx \equiv x='m' \vee x='f')$
$\forall xy(xHy \equiv Mx \ \& \ y='m' \vee Fx \ \& \ y='f')$

S2: $\forall x(Px \rightarrow$ Described x)
$\forall xy(xHy \rightarrow Px \ \& \ Gy)$
$\forall x(Gx \equiv x='m' \vee x='f')$
$\forall xyz(xHy \ \& \ xHz \rightarrow y=z)$
D2: $\forall x(Mx \equiv xH'm')$
$\forall x(Fx \equiv xH'f')$

The types String and Real are defined in all schemas. But the choice of Described object types is up to the designer. Each CS partitions the domain

of described objects into a set of primitive object types. So long as the domain D is the same, it is possible for schemas to specify different partitions but still be equivalent. Figure 6.21 illustrates two partitions, one demarcated by single lines and the other by double lines.

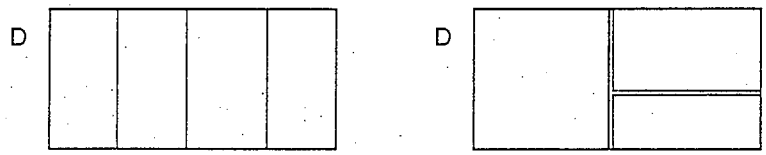


Figure 6.21 Two different partitions of the same domain

If two schemas differ in their described object types, then any claim of contextual equivalence must include definitions which enable these type predicates to be translated from one schema to the other. If the domains are not the same, this simply cannot be done. For example, consider the schemas shown in Figure 6.22. Both schemas have the object type Person but only CS2 has the object type Gender.

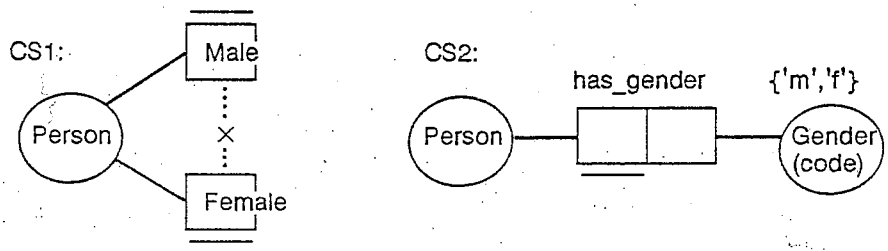
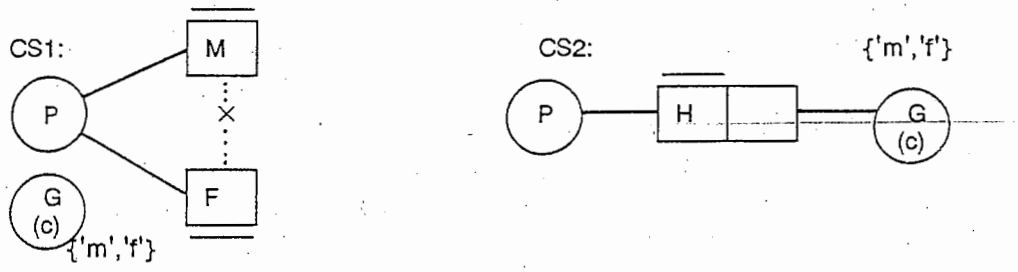


Figure 6.22 Schema CS2 has an extra described object type

Since the described object types differ it is necessary to consider their formalization. The solid ellipses translate via TSE to ∀x(Person x → Described x) and ∀x(Gender x → Described x). The global partition axioms P3 and P4 also make assertions about these types (sec. 5.2). Further, CS1 alone cannot accept some assertions that are acceptable for CS2, e.g. that there is a gender with code 'f'. Without access to global described object types which might make possible a translation of Gender, there is no way that CS2 can be converted into CS1.

In the absence of such global information, the only way that a contextual equivalence for such cases can be specified is to include all locally primitive, described object types and their reference schemes in both schemas. This is done in Figure 6.23. Here "H","G","c","C" abbreviate "has_gender", "Gender", "code", and "has_gendercode".



D1: ∀xy(xHy ≡ Mx & yC'm' ∨ Fx & yC'f')

D2: ∀x[Mx ≡ ∃y(xHy & yC'm')]
∀x[Fx ≡ ∃y(xHy & yC'f')]

Figure 6.23 Notice that G(c) {'m','f'} appears on both sides

Notice the need to specify the definitions using the reference schemes. To our knowledge, all previous treatments of this kind of equivalence have used individual constants to directly name the described objects (e.g. see NH89 p. 219). We now argue that this is a mistake, since this practice implies that the objects so named exist in all states of the knowledge base (since ICs must refer). Apart from other problems, this would make nonsense of the treatment of mandatory roles; for example, suppose we replaced "Gender (code) {'m','f'}" in the diagram by "Gender {m,f}", and made the second role of Has_gender mandatory. This entails that in each state of the knowledge base we must know the gender of at least one male and at least one female person: this is unacceptable. Notice that no such problem arises for lexical and numeric objects.

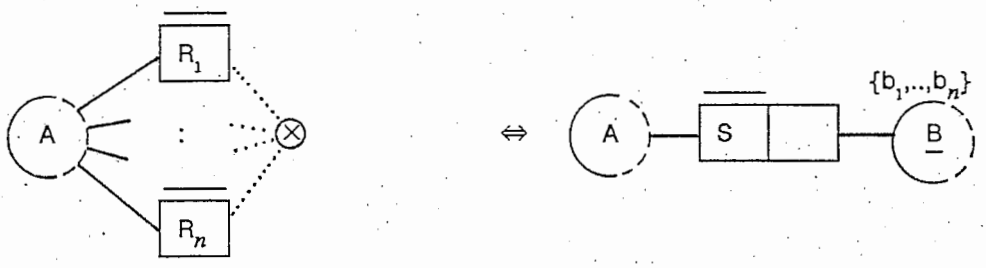
Although equivalence transformations must preserve described object types, in practice the designer may be satisfied with less than equivalence and decide to delete the old types. For example, suppose the binary in Figure 6.23 is replaced by the unaries and the Gender node. If Gender does not occur elsewhere in the global schema, and there is no need to support its form of expression, the designer may simply drop the Gender node as well as the definition D1. Again, we have controlled information loss.

Informal versions of the equivalences we have been discussing are called object-role reduction/composition by Falkenberg (1986). In NH89 we called them "entity type - fact type conversions", where the most important theorems were labelled "T2" and "T3" (NH89 pp. 222-3). We are now in a position to specify these theorems rigorously.

In this thesis all equivalence theorems have labels starting with "E". In each case CS1 & D1 ⇔ CS2 & D2 is understood as being asserted, where the left/right side is side 1/2. Because of space limitations, we do not provide further proofs. However, these may be proved by deduction trees in a similar manner to proof 5 in Appendix II. The unary-binary conversion for the

lexical/numeric case is set out in EUB1. Recall that when an object type must be either *lexical or numeric* it is shown as a half-solid circle with the type name underlined. Here we have n unary predicates, where n is any positive integer (in the trivial case $n = 1$, the exclusion constraint is vacuous).

EUB1

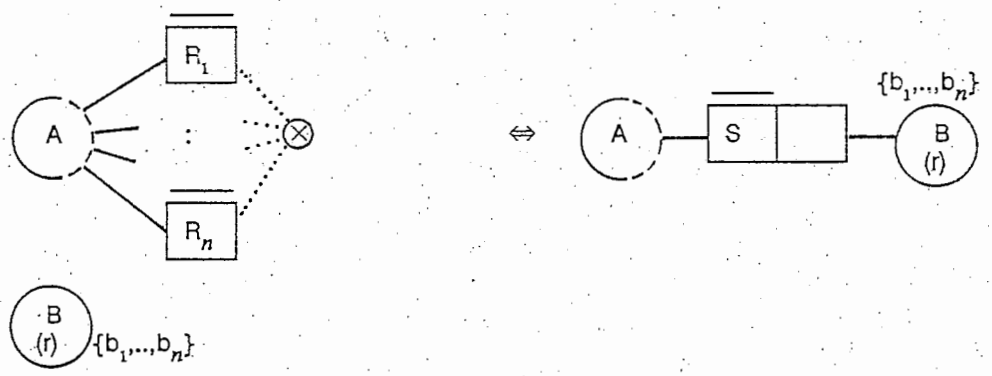


$\forall x(Bx \equiv x=b_1 \vee .. \vee x=b_n)$
$\forall xy(xSy \equiv R_1x \,\&\, y=b_1 \vee .. \vee R_nx \,\&\, y=b_n)$

$\forall x(R_1x \equiv xSb_1)$
:
$\forall x(R_nx \equiv xSb_n)$

The described object case is set out in EUB1'. In general the described object version of a theorem is named by priming the name of the corresponding lexical/numeric version. The members of $\{b_1,..,b_n\}$ are always lexical or numeric constants, and "$\approx_r$" denotes the reference predicate for the simple reference mode r.

EUB1'

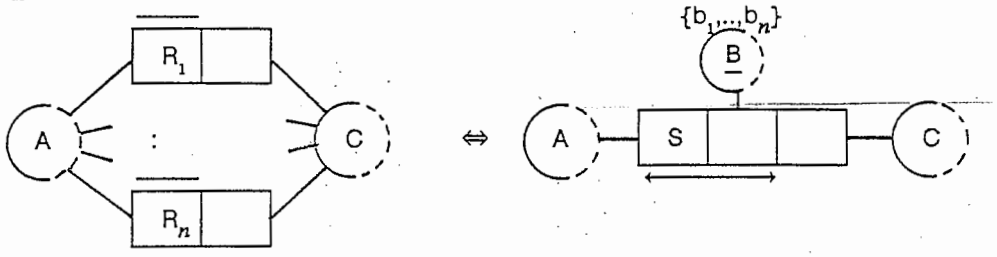


$\forall xy(xSy \equiv R_1x \,\&\, y\approx_r b_1 \vee .. \vee R_nx \,\&\, y\approx_r b_n)$

$\forall x[R_1x \equiv \exists y(xSy \,\&\, y\approx_r b_1)]$
:
$\forall x[R_nx \equiv \exists y(xSy \,\&\, y\approx_r b_n)]$

Clearly, if the exclusion constraint is deleted from CS1, the uniqueness constraint in CS2 must be deleted as well. We have deliberately avoided specifying any mandatory role constraint, so that the theorems have wider applicability. If the disjunction of $R_1.1$, .., $R_n.1$ is mandatory then of course so is S.1. Of more importance is the binary-ternary conversion (NH89 pp. 222-5). The basic theorems are EBT1 and EBT1':

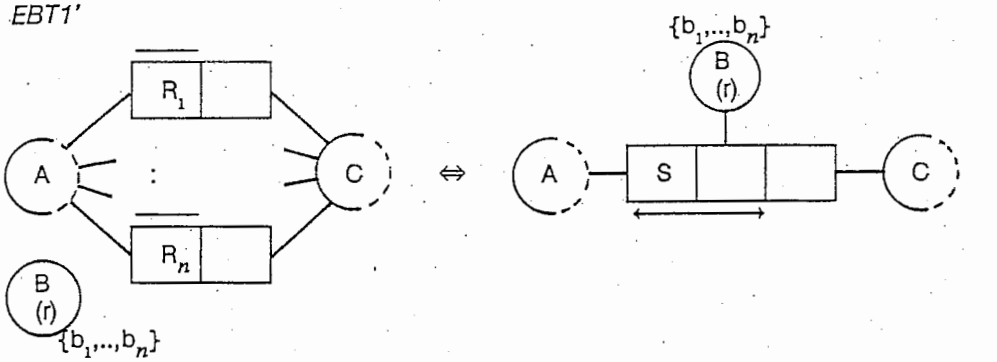
EBT1



$\forall x(Bx \equiv x=b_1 \vee .. \vee x=b_n)$
$\forall xyz(Sxyz \equiv xR_1z \,\&\, y=b_1 \vee .. \vee xR_nz \,\&\, y=b_n)$

$\forall xy(xR_1y \equiv Sxb_1y)$
:
$\forall xy(xR_ny \equiv Sxb_ny)$

EBT1'



$\forall xyz(Sxyz \equiv xR_1z \,\&\, y \approx_r b_1 \vee .. \vee xR_nz \,\&\, y \approx_r b_n)$

$\forall xy[xR_1y \equiv \exists z(Sxzy \,\&\, z \approx_r b_1)]$
:
$\forall xy[xR_ny \equiv \exists z(Sxzy \,\&\, z \approx_r b_n)]$

In NH89 (pp. 224-5) we introduced some new variations on the binary-ternary conversion. We now specify these for the *n*-ary case. To save space, diagrams are omitted.

EBT2 An occurrence frequency of *n* on S.1 in EBT1 or EBT1' is equivalent to an equality constraint between roles $R_1.1,..,R_n.n$.

EBT3 An occurrence frequency of *n* combined with a mandatory role constraint on S.1 in EBT1 or EBT1' is equivalent to mandatory role constraints on each of $R_1.1,..,R_n.n$.

As a prelude to specifying further variations, we examine a binary-ternary transformation presented by Falkenberg (1986). This example also illustrates how a formal approach can be used to clarify and check proposed equivalences needed for applications. Consider the two schemas of Figure 6.24. To simplify the discussion we have made Statuscode lexical, and shortened "supervisor" and "worker" to "s" and "w".

According to Falkenberg, the schemas in Figure 6.24 may be transformed into one another. However, even if definitions were supplied to translate

between the different predicate symbols, the constraint patterns shown are not equivalent. The uniqueness constraint in CS1 is not captured by CS2, since the latter permits an employee to be both a worker and a supervisor on the same project: what is required here is a pairwise exclusion constraint between the two binaries.

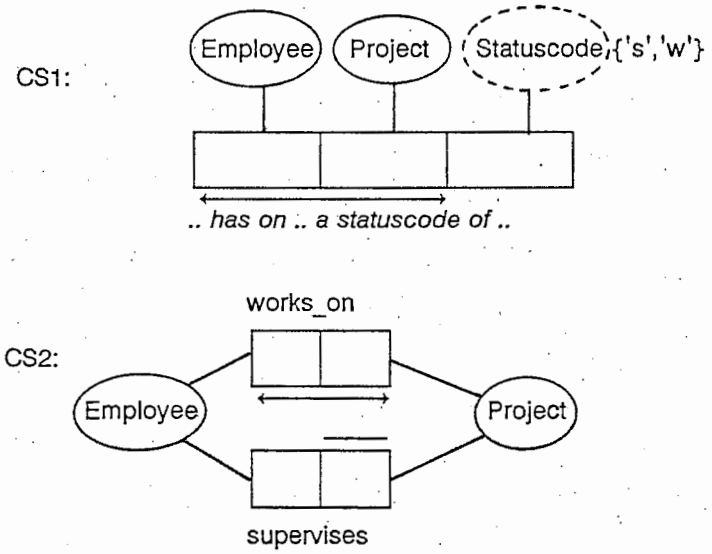


Figure 6.24 Under what conditions are CS1 and CS2 equivalent?

Moreover, the lower uniqueness constraint in CS2 is not captured by CS1: we need to add to CS1 the constraint that each project has at most one supervisor. There is no graphic notation in NIAM to do this, so we add it as a textual constraint in KL (a graphic notation could be devised, as suggested later). The contextual equivalence is set out in Figure 6.25, with predicate symbols shortened to one letter. Here we have separated the schemas from the contextual definitions by a broken line. Note that the textual constraint (below the diagram) is part of CS1, not D1.

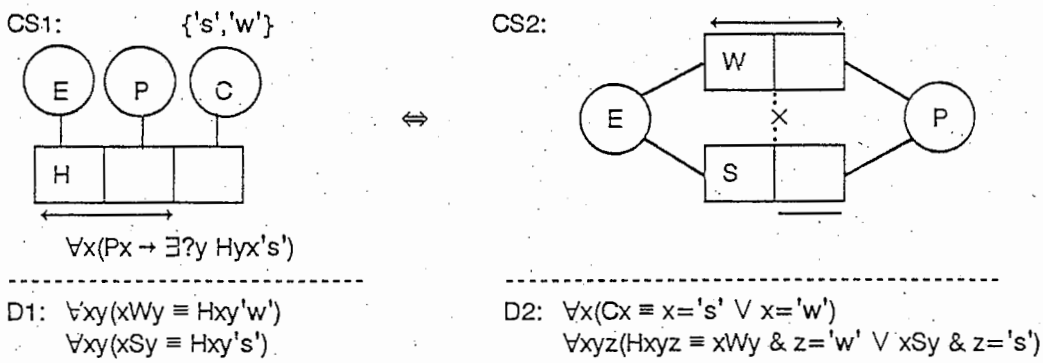
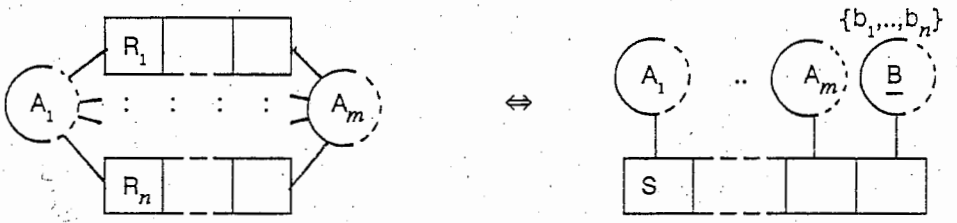


Figure 6.25 The conditions for equivalence are specified

Note the similarity between Figure 6.25 and EBT1. Each uses a type enumeration to juggle between shorter predicates and a longer predicate. The crucial differences are the constraint patterns. One systematic and efficient way to deal with such transformations is to first specify more general equivalences, and then specify the changes required when further constraints are added. EET1 and EET1' are general equivalence theorems which licence transformations between n predicates of arity m and a single predicate of arity $m+1$ one of whose object types is, or injects to, an enumerated set of n objects.

Although EET1-2 allow S to be transformed into a disjunction of smaller fact types, this does not entail that S is compound. A fact type is compound (non-elementary) iff each instance of it is equivalent to a conjunction of smaller facts. So S may still be elementary.

EET1

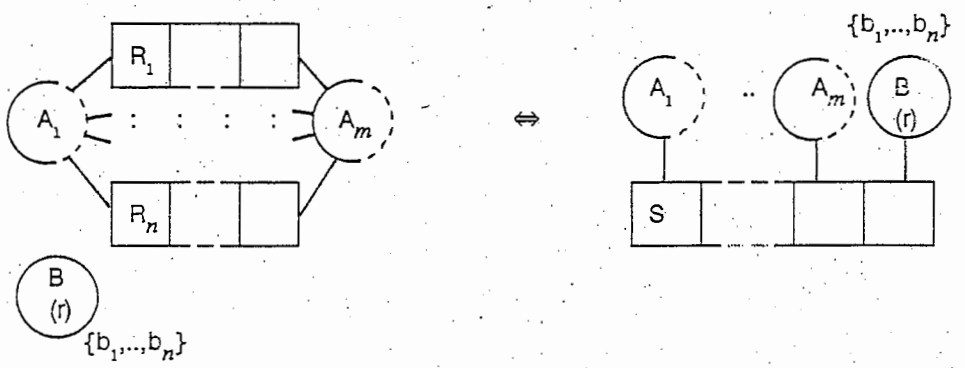


$\forall x(Bx \equiv x=b_1 \vee .. \vee x=b_n)$
$\forall \underline{x}\underline{y}(S\underline{x}\underline{y} \equiv R_1\underline{x} \,\&\, y=b_1 \vee .. \vee R_n\underline{x} \,\&\, y=b_n)$

$\{ \underline{x} = x_1..x_m \}$

$\forall \underline{x}(R_1\underline{x} \equiv S\underline{x}b_1)$
:
$\forall \underline{x}(R_n\underline{x} \equiv S\underline{x}b_n)$

EET1'



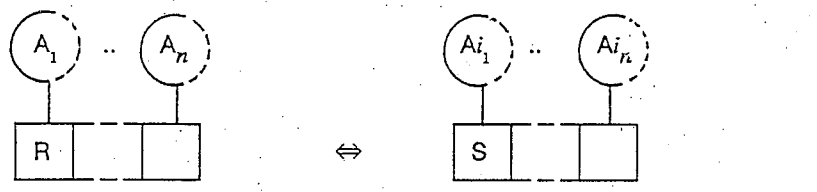
$\forall \underline{x}\underline{y}(S\underline{x}\underline{y} \equiv R_1\underline{x} \,\&\, y \approx_r b_1 \vee .. \vee R_n\underline{x} \,\&\, y \approx_r b_n)$

$\{ \underline{x} = x_1..x_m \}$

$\forall \underline{x}[R_1\underline{x} \equiv \exists y(S\underline{x}\underline{y} \,\&\, y \approx_r b_1)]$
:
$\forall \underline{x}[R_n\underline{x} \equiv \exists y(S\underline{x}\underline{y} \,\&\, y \approx_r b_n)]$

For convenience we specify theorems with the enumerated type B in a particular position. Analogous theorems apply when B is shifted to any other position. Theorem ERP allows roles to be permuted in any order by renaming the predicate. For example, we may wish to replace the fact type Person works_for Department by the fact type Department employs Person.

ERP



$\forall x_{i1}..x_{in}(Sx_{i1}..x_{in} \equiv Rx_1..x_n)$

$\forall x_1..x_n(Rx_1..x_n \equiv Sx_{i1}..x_{in})$
$\{ i_1..i_n$ is a permutation of $1..n \}$

Theorems EET1-1' made no mention of uniqueness constraints. Since a UC spanning all roles of a predicate adds no information, the theorems entail that a UC spans all of S if and only if UCs span each R_j, $j = 1$ to n. If a UC is stronger it must span all but one role (otherwise the fact type is compound). We divide these stronger situations up into three cases and specify the relevant theorems. In the first two cases, the role played by B is included in the scope of the UC. In the first case S is not a binary.

EET2 Corollary to EET1-1' where S is at least a ternary: if a UC is added which spans all but role i of S, where $i \leq m$, then for each R_j, $j = 1$ to n, a UC must be added spanning all but role i; and conversely.

The UC constraint mapping for EET2 is summarized in Figure 6.26. Here we take it that B either is $\{b_1,..,b_n\}$ or injects to this via some reference scheme. Given generalization by role permutation, it is clear that EBT1-1' are just special cases of EET2, where $m = 2$.

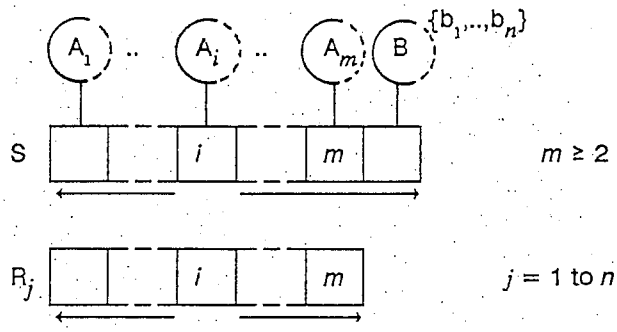
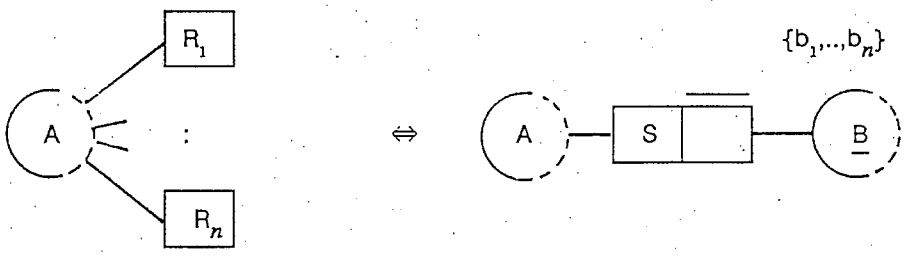


Figure 6.26 Mapping uniqueness constraints in theorem EET2

Consider the schemas of EET1-1' when $m = 1$ and a UC on S spans only the role played by B. If S maps each object in B to at most one object in A, each R_i can have at most one member. This leads to theorem EET3. Its simpler version (where B is lexical/numeric) is depicted in Figure 6.27.

EET3 Corollary to EET1-1' where S is binary: if a UC exactly spans the role played by B then $\#R_1 \leq 1$ & .. & $\#R_n \leq 1$; and conversely.



$\#R_1 \leq 1$ & .. & $\#R_n \leq 1$

$\forall x(Bx \equiv x=b_1 \vee .. \vee x=b_n)$
$\forall xy(xSy \equiv R_1x \& y=b_1 \vee .. \vee R_nx \& y=b_n)$

$\forall x(R_ix \equiv xSb_i)$, $i = 1$ to n

Figure 6.27 The simpler version of EET3

As an example of EET3 where B is described, recall the following n:1 binary fact type from section 6.2: Official (name) holds Position (code) {'P','S','T'}. Under the context specified, this may be transformed into a schema with three unary predicates (Is_president, Is_secretary, Is_treasurer) each of which has a cardinality limit of 1.

If, for the situation of EET3, the role S.1 is mandatory then the disjunction $R_1 \vee .. R_n$ is mandatory and $\#A \leq n$ (cf. implication theorem I#2').

The next theorem deals with the case where the UC spans all roles except the one played by B. The basic correspondence between the exclusion constraint in one schema and the UC in the other is shown in Figure 6.28. Each operand of the exclusion constraint is a whole predicate.

Theorem EET4 applies for all values of m. Clearly, EUB1-1' are just special cases of EET4 when $m = 1$.

EET4 Corollary to EET1-2: if a UC spans all roles of S except for B's role, then $R_1,..,R_n$ are mutually exclusive; and conversely.

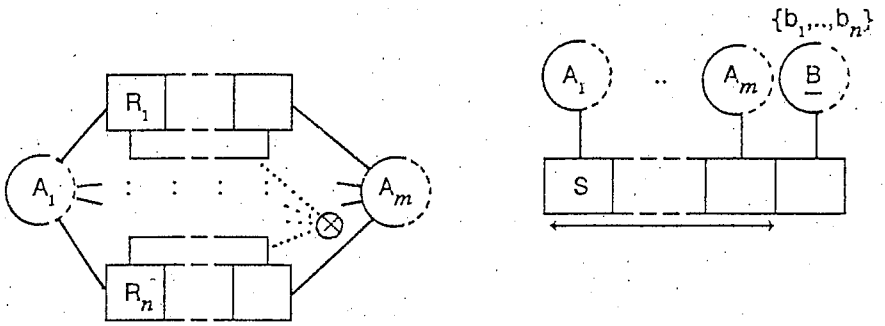
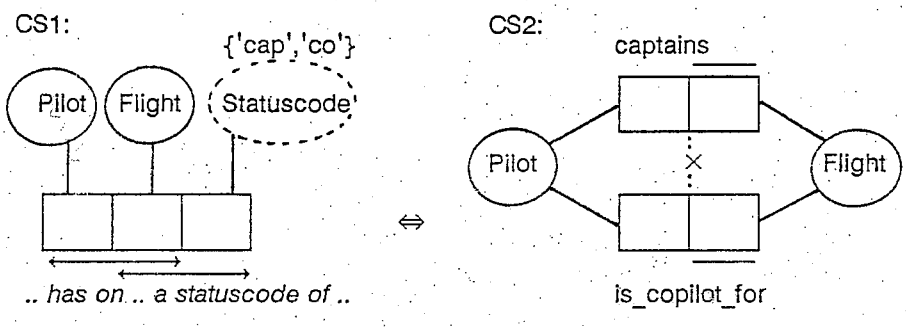


Figure 6.28 The UC in one schema corresponds to the × in the other

Figure 6.29 illustrates both EET2 and EET4 for the case $m = 2$. The definitions abbreviate the predicate symbols to their first letter. The leftmost UC maps to the exclusion constraint (EET4); the other UC in CS1 maps to the two simple UCs in CS2 (EET2).

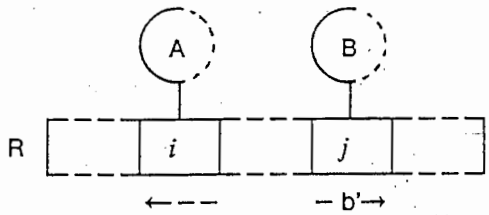


D1: ∀xy(xCy ≡ Hxy'cap')
∀xy(xIy ≡ Hxy'co')

D2: ∀x(Sx ≡ x='cap' ∨ x='co')
∀xyz(Hxyz ≡ xCy & z='cap' ∨ xIy & z='co')

Figure 6.29 Constraint mapping examples of EET2 and EET4

Another application of EET4 for the case $m = 2$ is the equivalence considered earlier in Figure 6.25; however we still have to specify a general theorem to cover the correspondence between its textual constraint and the simple UC. To facilitate this task, we first classify this kind of textual constraint. Suppose the *i*th and *j*th roles of predicate R are played by object types A and B. Then if the combination of each object in A with a given object b from B is unique within R, we say that there is a *restricted uniqueness constraint* on R between A and b. Such constraints may be specified textually (see the formula in Figure 6.30).



$\forall x[\, Ax \rightarrow \exists ?\underline{y}(R\underline{y} \,\&\, y_i = x \,\&\, y_j = b)\,]$

Figure 6.30 A restricted UC on R between A and b, where $b \in B$

As an aid to visualization, we have also shown this constraint graphically as a broken uniqueness bar including an "image" of b. This image b' is written the way a table entry for b would be written: if B is numeric then b' is b; if B is lexical then b' is b with the delimiting quotes deleted; if B is described and $b \approx_r e$ then the table entry form of e is used. Since NIAM already has many graphic notations, this new notation may be ignored in favour of the textual version. Note that if B is described and $b \approx_r e$ then the condition $y_j = b$ is expressed as $y_j \approx_r e$.

The textual constraint in Figure 6.25 is thus a restricted UC on H between P and 's' (this could be shown graphically as "⟵ – – s – –⟶" below the roles for P and C). The following theorem specifies how such constraints transform into UCs in the context of EET1-1'. The case where S is a binary may be ignored since it is already covered by EET1-1' (a full UC must exist between A and B).

EET5 Corollary to EET1-1': if S is at least a ternary, and a restricted UC is added between A_i and b, where b is or injects to b_j, then a UC must be added to R_j on the role played by A_i; and conversely.

If S is longer than a ternary, then R_j is splittable, since it has a simple key on $R_j.i$, and R_j is at least a ternary. A general requirement of the conceptual schema design procedure is that compound fact types be eliminated in favour of elementary fact types. Hence, in this case, R_j must be split on A_i into binaries with each of the other A's, with a simple UC on the role played by A_i in each of these binaries. We discuss splitting transformations shortly.

Note that the EET transformations cannot simply be adopted when the enumeration constraint $\{b_1,..,b_n\}$ is replaced by a cardinality constraint of 0;n on B. To develop analogous theorems for cardinality constraints we need to know that #typeB ≤ n, not just #B ≤ n. Moreover, we need to identify the

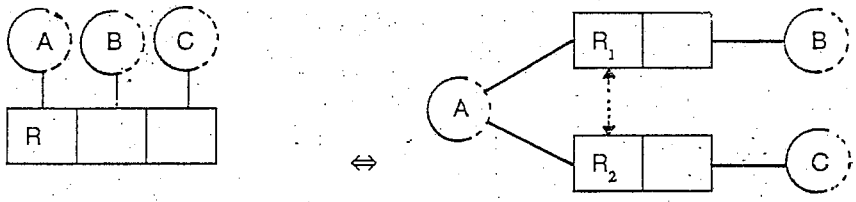
members of B to supply translations between the predicates in the different schemas. So in practice an enumeration is needed.

Sometimes, for the same predicate, more than one object type is enumerated (e.g. see NH89, p. 152, Figure 6.43). In this case the EET theorems may be applied more than once. Theorems EBT2-3 have obvious generalizations for this situation (note that when an *n*-item object type is eliminated here, the relevant frequency constraint is divided by *n*).

We now discuss a class of equivalence theorems (ESC1..) for *splitting* compound fact types or *combining* simpler fact types. The splitting and combining transformations basically correspond to the relational operations of projection and natural inner join respectively (e.g. see NH89, sections 4.3, 5.3). For conceptual schema design purposes, ESC equivalences are always used in the direction of splitting, since we aim to eliminate compound fact types. For relational database design, some recombination usually takes place.

A fact type is compound if and only if it can be expressed as a conjunction of simpler fact types. A fact type which is not compound is elementary. By far the most important case is deciding whether a *ternary* fact type is splittable. We first examine this case generally, and later consider the impact of functional dependencies. For a ternary predicate R over object types A, B and C (not necessarily distinct), we need to consider the possibility of splitting on A (i.e. splitting into two binaries over A-B and A-C), on B, on C, or into three binaries. ESC1 specifies the predicate translation context for splitting/combining on A.

ESC1

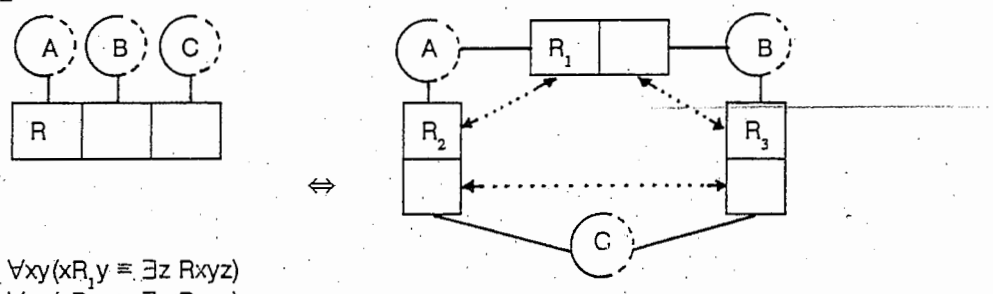


$\forall xy(xR_1y \equiv \exists z\, Rxyz)$
$\forall xy(xR_2y \equiv \exists z\, Rxzy)$

$\forall xyz(Rxyz \equiv xR_1y \;\&\; xR_2z)$

Notice the equality constraint. This is required since we do not allow null values at the conceptual level. The contextual translations are crucial: unless these hold, the splitting/combining cannot take place. Analogous theorems apply for splitting/combining on B or C. The 3-way split is specified in ESC2.

ESC2



$\forall xy(xR_1y \equiv \exists z\, Rxyz)$
$\forall xy(xR_2y \equiv \exists z\, Rxzy)$
$\forall xy(xR_3y \equiv \exists z\, Rzxy)$

$\forall xyz(Rxyz \equiv xR_1y \,\&\, xR_2z \,\&\, yR_3z)$

If a role is a simple key or functionally determines other roles then its object type is usually chosen as the node on which the splitting takes place. If this happens for more than one role, a choice exists; we do not consider such cases here. The next few theorems specify the impact of some additional FDs. To save space, diagrams are omitted. ESC3 and ESC5 are easily proved, and ESC4 follows immediately from ESC3 and IFD1.

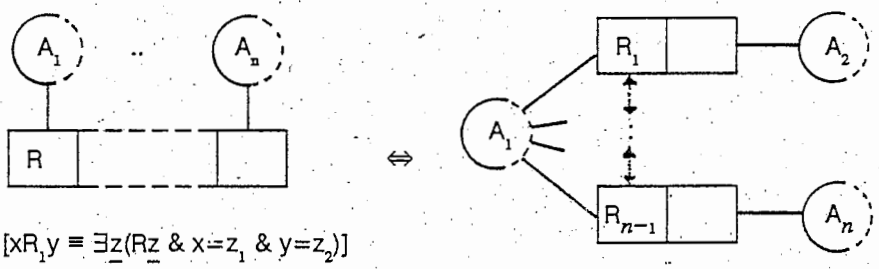
ESC3 Corollary to ESC1-2: if A functionally determines B (or C) in R, then A has a simple UC in R_1 (or R_2); and conversely.

ESC4 Corollary to ESC1-2: if A has a simple UC in R then it has a simple UC in R_1 and R_2; and conversely.

ESC5 Corollary to ESC1-2: if B and C form a composite key for R then a UC applies between $R_1.2$ and $R_2.2$; and conversely.

As the arity of a fact type increases, the number of potential ways in which it might be split increases rapidly. We specify only the most important case (n-ary fact type to/from n-1 binaries with common node).

ESC6



$\forall xy[xR_1y \equiv \exists \underline{z}(R\underline{z} \,\&\, x{=}z_1 \,\&\, y{=}z_2)]$
:
$\forall xy[xR_{n-1}y \equiv \exists \underline{z}(R\underline{z} \,\&\, x{=}z_1 \,\&\, y{=}z_n)]$

$\forall \underline{x}(R\underline{x} \equiv x_1R_1x_2 \,\&\, ..\, \&\, x_1R_{n-1}x_n)$

ESC6 depicts splitting on A_1. Analogous results hold for splitting on other nodes. Equality constraints are specified between adjacent first roles ($R_1.1$ and $R_2.1$, etc.). Since the identity relation (=) is transitive this implies that all first roles have identical populations.

The following theorems are generalizations of ESC3-4 to the n-ary case just considered. Theorems ESC6 and ESC8 are of special importance, since they are used in the proof of the ONF (optimal normal form) grouping algorithm.

ESC7 Corollary to ESC6: if A_1 functionally determines A_i in R then A_1 has a simple UC in R_{i-1}; and conversely.

ESC8 Corollary to ESC6: if A_1 has a simple UC in R then it has a simple UC in each of R_1, .., R_{n-1}; and conversely.

The research literature on the relational model of data includes numerous examples of splitting/combining transformations. In NIAM, these form one of the four classes of equivalences cited by Falkenberg (1986). While the formulation of the ESC theorems in terms of contextual equivalence is our own, the ESC theorems stated here have well known parallels in relational theory.

We now consider a class of schema implications and equivalences based on *frequency constraints*. Over the last few years we included examples of this class in our lecture notes and exercises. In an earlier paper (1988c) we referred to this class as "role disjunction". Though examples in this class often occur in practical applications, the only discussion besides our own that we have seen in print is by Leung (1988), who states, without proof, two results for the binary case. Developed independently, our approach differs from Leung's by being more general, and formally rigorous: though stated as equivalences, Leung's examples are actually just one-way implications.

Before specifying some general theorems, we consider a simple example (see Figure 6.31). In CS1 each employee has at most two phones. In CS2 each employee has at most one phone1 and at most one phone2 which must differ (if we wish to assert that an employee has a phone2 only if he/she has a phone1, a subset constraint should be added from has_phone2.1 to has_phone1.1). Can an equivalence context be specified for transforming between CS1 and CS2?

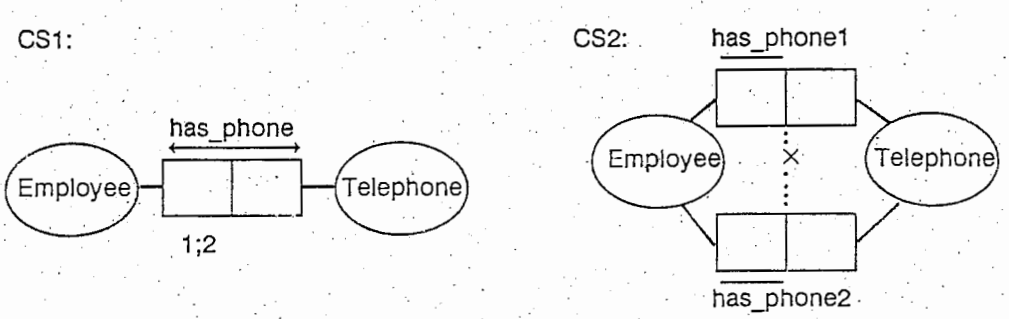
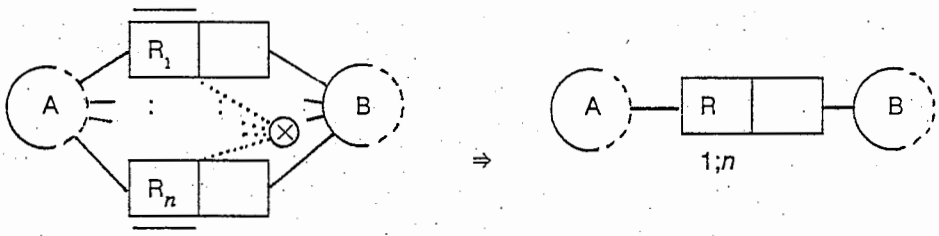


Figure 6.31 Can an equivalence context be specified?

CS1 may be defined in terms of CS2 by adding D2: ∀xy(x has_phone y ≡ x has_phone1 y ∨ x has_phone2 y). But there is no feature of CS1 which can provide a definition D1 of the predicates in CS2. In this sense, CS2 is *"stronger"* or more informative than CS1. Similarly, the captain-copilot schema in Figure 6.29 is stronger than a schema with a single binary saying that a flight has at most two pilots. The stronger schemas have the added advantage of having simple UCs, which enables the information to be stored in the same table with other attributes of the key.

In such cases the designer will often choose the stronger schema. We now set out some relevant schema implication theorems. To distinguish schema implication theorems from constraint implication theorems, the names of the former start with "Im".

ImFC1

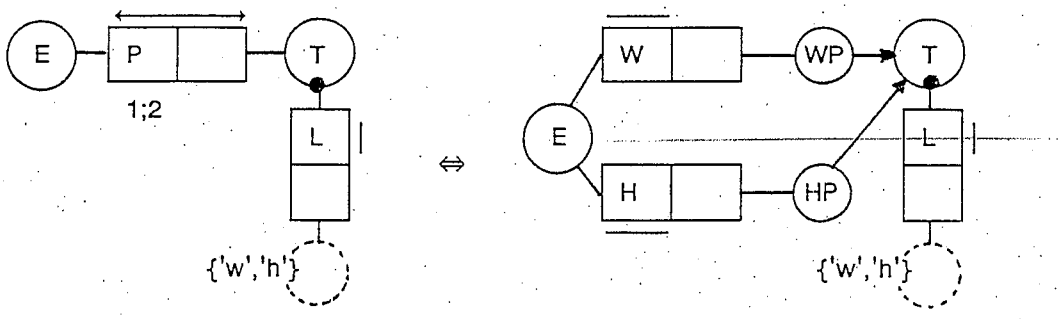


$\forall xy(xRy \equiv xR_1y \vee .. \vee xR_ny)$

ImFC2 Corollary to ImFC1: if an equality constraint applies among roles $R_1.1,..,R_n.1$ then R.1 has a frequency constraint of *n*.

ImFC3 Corollary to ImFC1: if roles $R_1.2,..,R_n.2$ are mutually exclusive and each has a simple UC, then R.2 has a simple UC.

Sometimes an equivalence, rather than just an implication result, can be obtained by using additional context which enables each of the disjuncts to be defined in terms of the other schema. For example, let us modify the situation of Figure 6.31 so that globally, each telephone has exactly one location-code which classifies it as a work or home phone. An equivalence may now be specified as shown in Figure 6.32. From IX2 the subtypes are exclusive and so are the roles W.2 and H.2: we omit these exclusion constraints since they are implied. The textual constraint says that no employee has two phones with the same location code. The predicate dictionary is: E = Employee, T = Telephone; WP = Workphone; HP = Homephone; P = has_phone; W = has_workphone; H = has_homephone; L = has_locationcode.



{ No emp. has 2 phones with the same locationcode }
∀xyzuv(xPy & xPz & y≠z & yLu & zLv → u≠v)

∀x(WP x ≡ xL'w')
∀x(HP x ≡ xL'h')
∀xy(xWy ≡ xPy & yL'w')
∀xy(xHy ≡ xPy & yL'h')

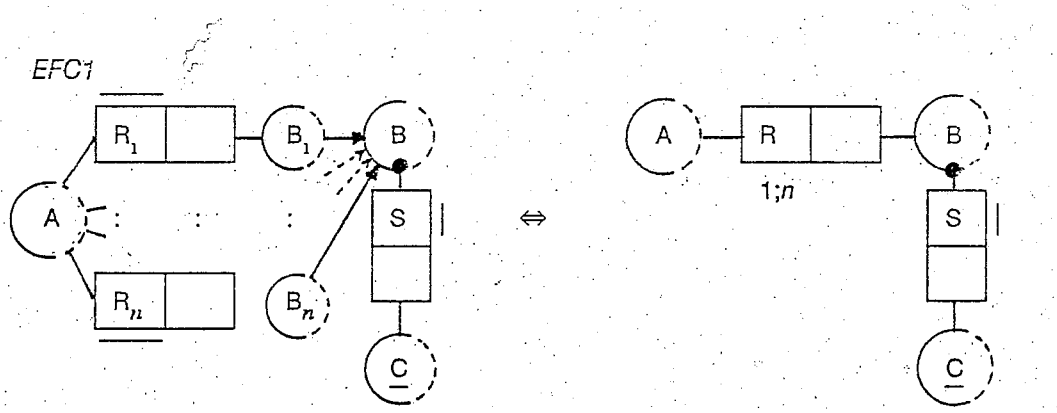
∀x(WP x ≡ xL'w')
∀x(HP x ≡ xL'h')

∀xy(xPy ≡ xWy ∨ xHy)

Figure 6.32 The additional context supports equivalence

In practice, the structure of the additional context required for equivalence depends on the particular case, though the basic form in this example is the most common. For simplicity, we specify our equivalence theorems for those cases where the context can be provided as simply as in this example; described object types may be catered for in the usual way (we give an example in the next chapter). The constants $c_1,..,c_n$ are lexical or numeric. If the additional context is not required globally, the designer may deliberately weaken the schema by deleting this context.

EFC1



∀x(B$_i$x ≡ xSc$_i$), i = 1 to n

∀xy(xRy ≡ xR$_1$y ∨ .. ∨ xR$_n$y)

∀xyzuv(xRy & xRz & y≠z & ySu & zSv → u≠v)

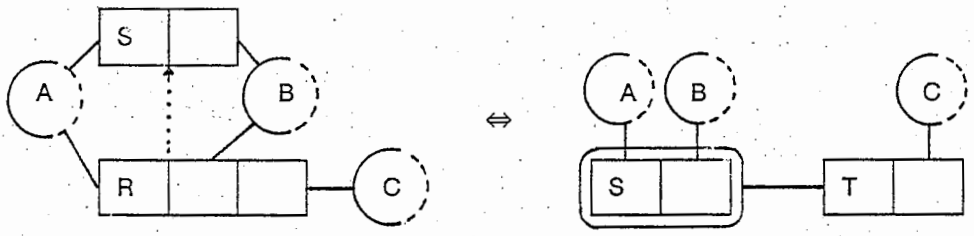
∀x(B$_i$x ≡ xSc$_i$), i = 1 to n
∀x(xR$_i$y ≡ xRy & ySc$_i$), i = 1 to n

EFC2 Corollary to EFC1: if an equality constraint applies among roles $R_1.1,..,R_n.1$ then R.1 has a frequency constraint of n; and conversely.

EFC3 Corollary to EFC1: if roles $R_1.2,..,R_n.2$ each has a simple UC, then R.2 has a simple UC; and conversely.

We now briefly discuss *nesting/flattening* transformations. In NH89 (section 10.3) we extended the traditional treatment of these transformations by considering the impact of mandatory roles, and developing an algorithm based on the degree of overlap of participating pair types. We have space here only to discuss a couple of examples. In the next chapter we discuss some applications of the overlap algorithm. The following theorems are among the more important in this class, and should be sufficient to illustrate the relevant formalization issues.

ENF1



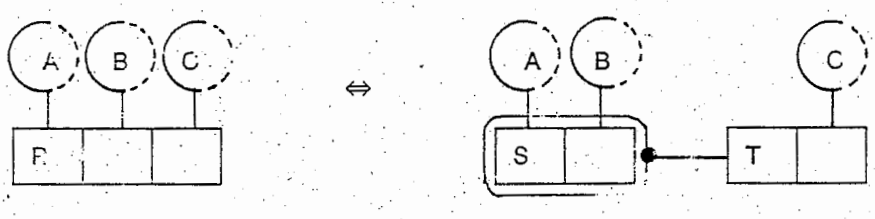
$\forall xy[xTy \equiv \exists zw(x = (z,w) \ \& \ Rzwy)]$ $\forall xyz(Rxyz \equiv (x,y)Tz)$

In our formalization, since the pair function is generic there is no need to introduce a special function or predicate to handle specific cases of nesting, and hence no separate clause is needed to translate the embedded part in the flattened version.

ENF2 Corollary to ENF1: if an equality constraint exists between S and R.1-2 then the role T.1 is mandatory; and conversely.

If it is not desired to actually store S in the flattened version, then it may be specified instead as a derivation rule. This is set out as ENF2*. Compare this with ENF2.

*ENF2**



$\forall xy[xTy \equiv \exists zw(x = (z,w) \ \& \ Rzwy)]$ $\forall xyz(Rxyz \equiv (x,y)Tz)$
$\forall xy(xSy \equiv \exists z \ Rxyz)$

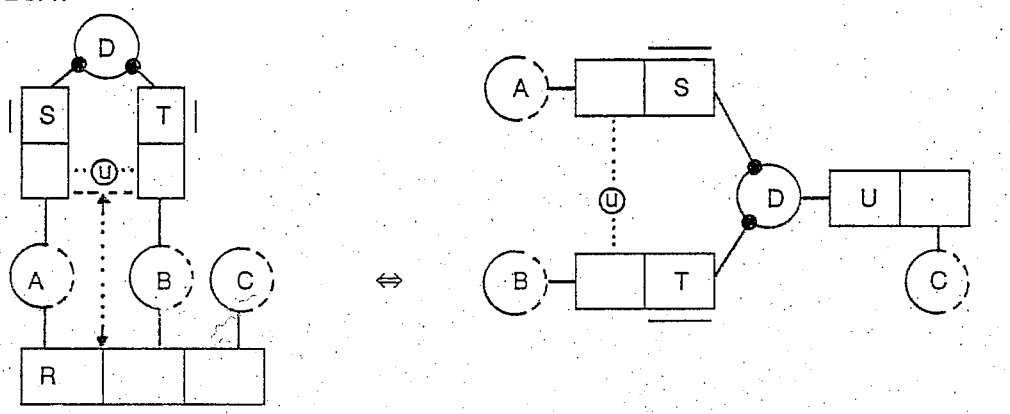
The next theorem discusses the impact of uniqueness constraints.

ENF3 Corollary to ENF1 and ENF2*: if a UC exactly spans R.1-2 then a simple UC spans T.1; and conversely.

Further nesting/flattening theorems and several examples are discussed in NH89 (section 10.3). As discussed later, we generally prefer the flattened version unless the pair type plays more than one role or at least one optional role.

The last of the four traditional NIAM schema transformations deals with the use of a *compositely described object type* (see NH89, section 10.4). This is analogous to nesting/flattening for the mandatory role case. The following two theorems in this category are representative. For naming purposes, we now include these within a more general class of transformations we call *object type addition/deletion.*

EOA1



∀xy[xUy ≡ ∃zw(xSz & xTw & Rzwy)] ∀xyz[Rxyz ≡ ∃w(wSx & wTy & wUz)]

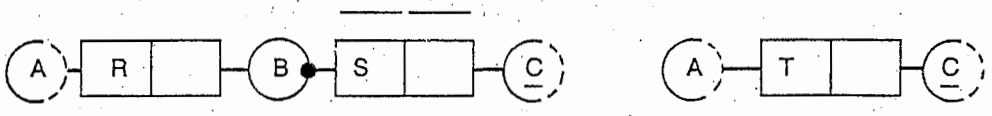
In practice, if the left-hand option is chosen, D is deleted unless there is a good reason for retaining it (e.g. if D plays other roles globally).

EOA2 Corollary to EOA1: if a UC exactly spans R.1-2 then a simple UC spans U.1; and conversely.

Object types may be introduced simply because the designer finds it conceptually easier to think in terms of them, or because they play other roles in the global schema. This may even happen with simply described types. For example, the fact that Reagan has a gender with code 'm' could be handled as a simple binary between Person and Gendercode. But usually one wishes to think in terms of Gender: in this case the information is handled as two

binaries (one fact and one reference). Introducing a new described object type leads to a stronger schema, as indicated in the following schema implication theorem.

ImOA1



$\forall xy[xTy \equiv \exists z(xRz \,\&\, zSy)]$

We briefly note another class of schema equivalences dealing with *choice of derived predicate*, according to the degrees of freedom available. For example, the schemas in Figures 5.24 and 5.25 are equivalent. What is perhaps surprising is that the alternative derivation rules are not equivalent by themselves. To obtain equivalence, the complete constraint patterns shown in these schemas must be included. Though not included in this thesis, the formal proof (by deduction tree) is instructive in highlighting the part played by each constraint to produce the equivalence.

Another practical example of this kind is discussed in NH89 (p. 168). The basic schema is set out in Figure 6.33. Here a subject may be identified either by its subject code (e.g. "CS112") or by the combination of its discipline, level and serial#. However the subject code is just the concatenation of the disciplinecode, levelcode and serial#. So one has the option of making the subjectcode predicate derived or of making all the other three predicates derived (mandatory role dots on the derived roles are implied and are best omitted). Conceptually, either choice may be taken, and our formalization of String operations and reference enables the derivation rules to be specified without difficulty. The two different schemas resulting can be formally proved to be equivalent.

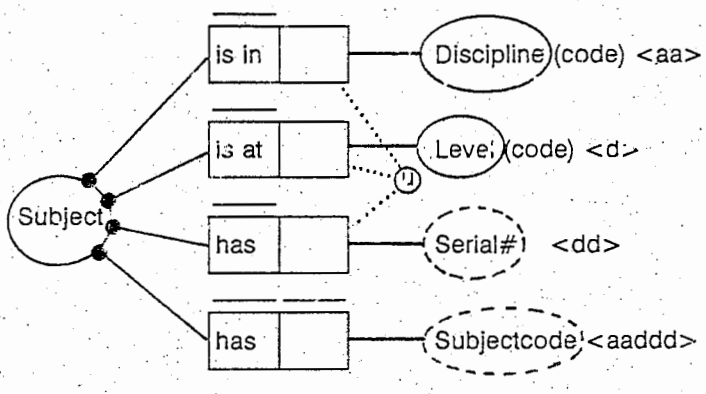
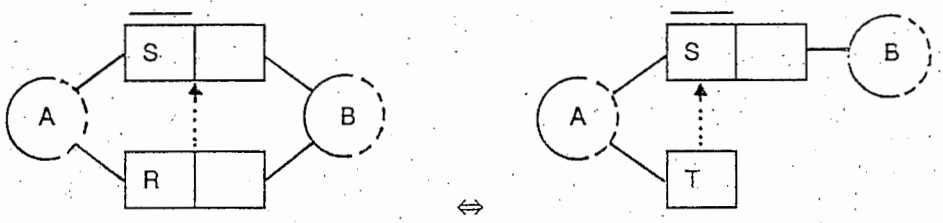


Figure 6.33 A basis for two choices of primary reference scheme

One new equivalence theorem that we will make use of in the next chapter is set out as ESS1. It deals with the case where binary predicates with simple keys are bound by a pairwise subset constraint. From IFD2 and IU2 it follows that R.1 is a simple key. The rest of the proof is obvious. In practice the transformation from left to right is usually preferred.

ESS1



$\forall x(Tx \equiv \exists y\, xRy)$ $\forall xy(xRy \equiv xSy \,\&\, Tx)$

Other equivalence theorems have been developed (e.g. to transform between figures 5.15 and 5.16), but there is no space in this thesis for a full account of our work in this regard. The main contribution of this section has been the provision of a rigorous foundation for specifying and proving theorems concerning schema equivalence and implication.

7 Some applications to relational database systems

7.1 The ONF algorithm: constraint mapping

In this chapter we discuss some practical uses of conceptual modelling in the formulation of schemas to be implemented in relational database systems. This section indicates how conceptual constraints may be mapped onto relational constraints. The next section illustrates various ways in which the conceptual schema may be optimized before the relational mapping takes place. In the final section, a global optimization procedure is outlined for selecting transformations to apply to a global conceptual schema.

A *relational schema* specifies the UoD in terms of constructs supported by relational database systems. We assume the reader is familiar with relational database systems, and SQL in particular. Relational schemas differ from NIAM conceptual schemas in several ways. In a relational schema, fact types are often compound, and may include null values. The roles of these predicates (table types) are given attribute (column) names. Object types are called domains, but few systems provide support for domains other than lexical, numeric, date and time. General subtyping is not directly supported, but subtype constraints may be coded as procedures.

The distinction between stored and derived fact types is important for relational systems. Stored fact types (base tables) may be used for updates and queries. Derived fact types coded as procedures can be used only for queries. Some derived fact types may be defined as views (virtual tables): these may be queried but only in simple cases can these be updated (e.g. views defined in terms of a join are non-updatable). Some constraints may be enforced within base table and index definitions, but many constraints need to be coded as procedures. Typically, relational database systems do not support recursion, so recursively derived fact types (e.g. ancestorhood, parts explosion) are usually handled by embedding the system within a recursive language (e.g. Prolog).

In contrast to traditional normalization approaches, NIAM first produces a conceptual schema and then maps this down onto a relational schema. In NH89

(pp. 247-273) we discussed the mapping process in detail; we have space here only for a brief treatment, focussing on constraint mapping.

Although not discussed here, a rigorous treatment of this mapping may be developed within our formalization by extending our conceptual framework to cater for null values in the relational sense. Basically this can be done by allowing the null object to play any non-referential conceptual role, and providing appropriate semantics.

In NIAM, a conceptual schema has its stored fact types grouped into relational base tables via the *ONF (optimal normal form) algorithm*. Each fact type is grouped into only one table (hence no redundancy). Composite keys map to separate tables. Simple keys attached to the same object type are grouped into the same table, keyed on the object type identifier. Each remaining fact type maps onto a separate table. With 1:1 fact types a choice is made favouring fewer null values. Subtypes are absorbed into their primitive supertypes before mapping. We make no claim to originating this algorithm, which has long been used within NIAM and other methodologies.

Provided the conceptual fact types are elementary, the tables produced by the ONF algorithm are in 5th normal form. The term "optimal normal form" was used by Nijssen to emphasize that not only are the tables in 5NF but also some optimization has been achieved (e.g. a effort has been made to minimize the number of tables). The algorithm can be refined further to result in fewer tables for certain 1:1 cases, but we do not discuss this refinement in this thesis.

The ONF algorithm applies to global schemas. It also applies to subschemas so long as no object type in the subschema has an additional simple key attached in the global schema. While the ONF algorithm provides a simple, safe and reasonably efficient means of grouping fact types into tables, it fails to specify how constraints are mapped (except for keys). Apart from our own work, we know of no analysis which augments the ONF algorithm by comprehensive constraint mapping. Various aspects of constraint mapping were introduced in NH89. We now summarize and expand on this work.

In specifying relational schemas, we use a shorthand notation which includes constraint markers. Many of the conceptual notations are used or adapted; in some cases new notations are used. An intra-table uniqueness constraint is shown by underlining the names of columns spanned by the UC; arrowheads are added if the columns are not contiguous. If there is more than one candidate key, the primary key is doubly underlined. Equality constraints are shown as dotted lines without arrowheads.

While most NIAM constraints map directly to relational constraints, special treatment is required for mandatory roles since object types are not directly supported. A column is assumed *mandatory to its table* (i.e. not null) unless an optional marker "OP" is appended. Mandatory and optional roles of the same object type that map to columns of the *same table* are declared mandatory and optional, respectively, for that table. When two roles played by the same object type map to columns of *different tables*, proceed as follows: if both roles are mandatory, specify an equality constraint between their target columns; if only one role is optional specify a subset constraint from its target column to the target column of the other.

To illustrate the basic idea of constraint mapping, consider the output report shown in Table 7.1. Here the codes "C" and "S" denote the possible coal kinds (coking and steaming), and "?" denotes a null value, indicating that for some reason or other an actual value is not recorded.

mine	country	yr_opened	coal kind	reserves (Mt)
Lucky	USA	1985	C	120
			S	120
Rocky	USA	?	C	395
			S	0
Newie	UK	?	?	?

Table 7.1 Extract from a report about coal mines

Assuming the population of this table is significant, one way of conceptually schematizing this is shown in Figure 7.1.

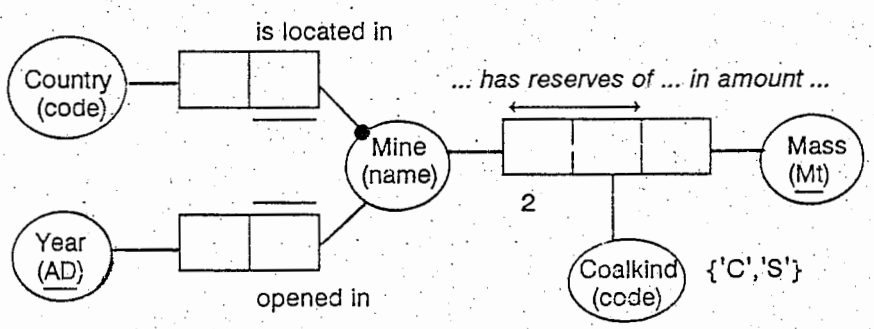


Figure 7.1 A conceptual schema for Table 7.1

Using the ONF algorithm, the three conceptual fact types map into two tables as shown in Figure 7.2, with keys as indicated. The mandatory role constraint is captured by the (default) declaration that columns mine and country are mandatory for table Coalmine, together with the subset constraint from CoalReserves.mine to Coalmine.mine.

Notice also that the frequency constraint of 2 and the enumeration constraint {'C','S'} carry directly across. Though not shown here, the syntactic data types for the columns must be specified to be consistent with the reference schemes: so mine and country are lexical (e.g. varchar) and yr_opened and reserves are numeric (e.g. smallint). In NH89 we gave examples of specifying syntactic domain constraints below the column names (e.g. a conceptual constraint of <c20> on Countrycode may be written as vc20, meaning varchar(20)).

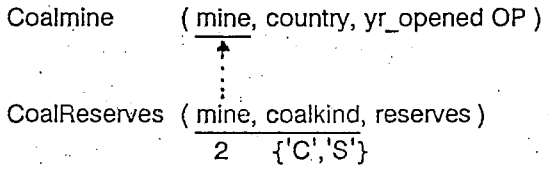


Figure 7.2 A relational schema for Table 7.1

Once the relational schema has been specified in this way, it is translated into code (table definitions, procedures etc.) understood by the particular database management system (DBMS). Both the mapping from the conceptual to the abbreviated relational schema, and the subsequent mapping to the DBMS code can be fully automated. However, the latter mapping depends to varying degrees on the target relational DBMS. Figure 7.3 illustrates the mapping of the schema in Figure 7.2 to an SQL system based on extended ANSI SQL (see Date 1987, ch. 11), assuming appropriate syntactic domains have been specified.

The code for the frequency constraint assertion is not standard, but this kind of facility is provided in some versions of SQL. In basic SQL the constraint can be enforced by storing the embedded **select** command as a routine and running this each time an update is attempted (with autocommit off). For efficiency reasons, if many elementary updates involving such constraints are issued, they are best processed as a bulk transaction.

Many current SQL systems do not even support the ANSI extensions used here. With such systems the primary key constraint can be enforced either by creating a unique index or by running a query to check for entry groups having **count**(*) above 1. The subset constraint expressed in the **references** clause can also be enforced by checking that no rows are returned by the following query: **select** mine **from** CoalReserves where mine **not in** (**select** mine **from** Coalmine). This is an example of "referential integrity". A similar approach may be used to code equality and exclusion constraints.

```
create table Coalmine
  ( mine        varchar(20) not null   primary key,
    country     varchar(20) not null,
    yr_opened   smallint  )

create table CoalReserves
  ( mine        varchar(20) not null   references Coalmine,
    coalkind    char        not null
                check ( Coalreserves.coalkind in ('C','S') ),
    reserves    smallint    not null,
    primary key (mine, coalkind) )

on insert, delete, update of CoalReserves
    if exists
      (select mine from CoalReserves
       group by mine
       having count(*) <> 2)
    then cancel,
         print 'Rejected: each mine must occur exactly 2 times'
```

Figure 7.3 Extended SQL code for the schema of Figure 7.2

Intra-table uniqueness constraints other than those on a primary key are specified in extended ANSI SQL at table creation time by prepending the keyword **unique** to the column list spanned by the UC. Alternatively, the unique index or group count approach may be used. Inter-table uniqueness constraints may be enforced by checking that no rows are returned when the tables are joined and the relevant group count is specified to be above 1. An example from the next section includes a UC between Assigned.subject and Instructs.student where the join attribute is lecturer: this UC may be coded by checking that the following query returns the null set.

```
select student, subject from Assigned, Instructs
where Assigned.lecturer = Instructs.lecturer
group by student, subject having count(*) > 1
```

The enforcement of irreflexivity is very efficient since this is a "restriction predicate", i.e. the constraint can be tested for a given row by examining that row alone. Hence, in extended ANSI SQL this may be specified as a check clause. For example, if the relation Examines (examiner, candidate), where the columns are defined over the same domain (say Academic), is irreflexive this constraint could be coded as:

```
check ( Examines.examiner <> Examines.candidate )
```

In basic SQL, this constraint may be enforced by checking for a null return from: **select** * **from** Examines **where** examiner = candidate.

Asymmetry and intransitivity constraints may be enforced using correlated subqueries. For example, the relation Father_of (father, child) is both asymmetric and intransitive: these constraints may respectively be enforced by checking that the following queries return the null set:

```
{ asymmetry check }
select * from Father_of X
where exists
  (select * from Father_of
   where father = X.child and child = X.father)

{ intransitivity check }
select * from Father_of X
where exists
  (select * from Father_of Y
   where father = X.child and exists
     (select * from Father_of
      where father = X.father and child = Y.child))
```

The basic approach of checking for a null return from the relevant query can be used to code the relational version of every NIAM conceptual constraint, including subtyping, that we have discussed. The coding is straightforward, but we discuss no further cases here.

Database design workbenches are available which produce relational table definitions from higher level specifications. Some automated design aids even generate code for a number of basic constraint types, including referential integrity (e.g. see Casanova & Tucherman 1988). However, we know of no software product which performs a mapping of NIAM constraints to the extent discussed here. We plan to implement such a mapper in the near future.

7.2 Conceptual schema optimization

Recent work by Falkenberg (1988) argues for deterministic modelling, in which each given UoD has only one "correct" conceptual schema. While our approach does include default guidelines for selecting schema transformations, it still leaves a reasonable amount of freedom to the designer in modelling the UoD (cf. Kent 1982). In this section we discuss the notion of "optimizing" a conceptual schema before applying the ONF mapping. The next section outlines a procedure for optimizing a global schema. It is beyond the scope of this thesis to provide a complete treatment of this topic.

We allow that the same UoD may be portrayed by different, but equivalent, conceptual schemas. These may produce different relational schemas when the ONF mapping is performed. Not only the width of the tables but also the number of tables may differ. For a given application or expected query/update pattern, one of these relational schemas will be most suitable. One could start with any of these relational schemas and then perform relational transformations to improve the efficiency.

However, rather than perform all the optimization at the relational level, we argue that it is better to perform *preliminary optimization at the conceptual level* by transforming the conceptual schema into a version which yields a better ONF map. The higher level semantics are easier to work with, the restriction to elementary fact types simplifies the transformation steps, and the ONF algorithm is kept simple. Of course, further optimization may be applied later (e.g. tuning with controlled redundancy). A somewhat similar approach in the context of ER modelling has been recommended by Teorey, Yang and Fry (1986 p. 220).

Thus, although optimization is not strictly an issue at the conceptual level, once it is decided that a relational DBMS will be used and an overview of the expected query/update pattern is available, there is much to be gained by optimizing the conceptual schema for this situation. Note that "optimization" is a relative term: it is always possible to find a query/update pattern to make any schema from a class of equivalent schemas the most efficient for that pattern.

However, our default guidelines are designed to favour conceptual schemas which provide an ONF map with *fewer tables*. This reduces the number of potential table joins, thus leading to faster data retrieval for, as well as simplifying the formulation of, queries which would have otherwise required additional joins. In addition, updates which formerly involved inter-table constraints may be facilitated by conversion to intra-table constraints.

We gave one example of conceptual optimization in NH89 (pp. 226-7). As a further example, consider the schema of Figure 7.1 (see previous section). The composite key on the ternary requires a separate table in the ONF map. Because of the enumerated type constraint on Coalkind we can replace this ternary with two binaries. Using theorems EBT2 and EBT3 we obtain the alternative conceptual schema shown in Figure 7.4.

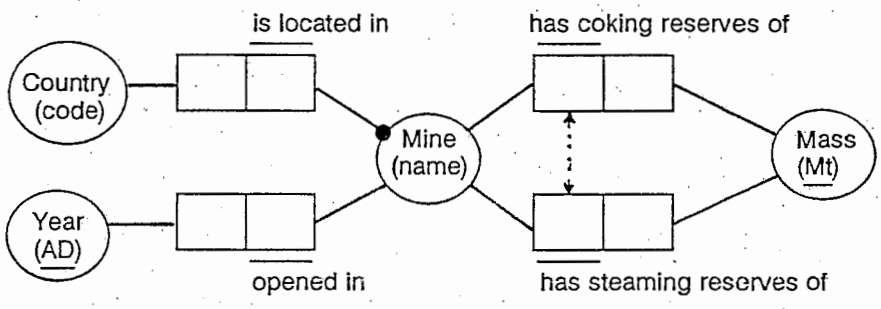


Figure 7.4 An optimized conceptual schema for Table 7.1

Instead of a composite key we now have two simple keys attached to an object type (Mine) which already has a simple key attached: so the new binaries can be grouped into the table which records attributes of Mine. When input to the ONF algorithm this results in the relational schema of Figure 7.5. The equality constraint line between the "OP"s indicates equality between the null value patterns.

Coalmine (mine, country, yr_opened OP, c_reserves OP, s_reserves OP)

Figure 7.5 The relational schema obtained from Figure 7.4

For most situations the relational schema of Figure 7.5 is more efficient than that of Figure 7.2. Instead of two tables we have just one table. So queries that formerly required a join of two tables can now be specified in terms of a single table. As a default guideline we select an equivalence transform (or an implication transform if we agree to the specified loss or gain of information) if we thereby replace a *unobjectified* composite key with simple keys attached to the same object type. This selection criterion becomes stronger if the object type already has other simple keys attached; if this is not the case, we at least simplify some queries which would have required a self-join.

Note that the case where the composite key applies to an objectified tuple type was excluded from the previous guideline. We now discuss an example which illustrates the reason for this qualification, as well certain other issues. Sometimes, using an equivalence to transform a subschema produces an object type which is already present in the global schema. In this case, care is required to avoid duplicating the object type on the schema diagram and to ensure its mandatory role constraints are specified correctly.

Consider the UoD depicted by the schema of Figure 7.6. Here animals are classified as herbivores, carnivores or omnivores (h, c or o) according as they eat just plants, just meat or both. For each zoo, the number of animals in stock must be recorded. Optionally, for any given zoo and animal kind the relevant food bill may be recorded.

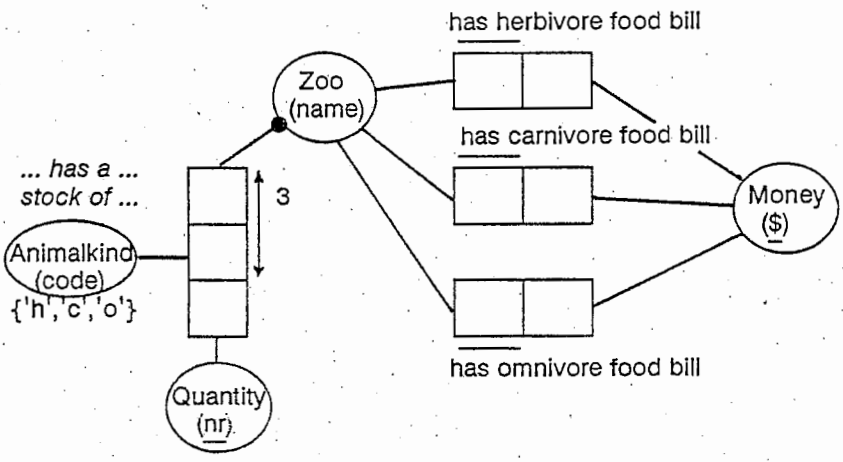


Figure 7.6 A conceptual schema about zoo animals

This schema has an ONF map with two tables, as shown:

```
                3     {h,c,o}
Stock   ( zoo, animalkind, quantity )
          ↑
          ⋮
Food    ( zoo, h_foodbill OP, c_foodbill OP, o_foodbill OP )
```

One way of improving it is to use theorems EBT2 and EBT4 to replace the ternary with three mandatory binaries with simple keys: Zoo has herbivore stock of Qty; Zoo has carnivore stock of Qty; Zoo has omnivore stock of Qty Zoo then has six simple keys attached, so the revised conceptual schema maps to just the following ONF table:

```
Zoostock ( zoo, herbivore_qty, carnivore_qty, omnivore_qty,
           herbiv_foodbill OP, carniv_foodbill OP, omniv_foodbill OP )
```

However, an alternative optimization strategy may be used. Using theorem EBT1', the three binaries of Figure 7.6 may be replaced by the ternary: Zoo has Animalkind food bill of Money (see Figure 7.8).

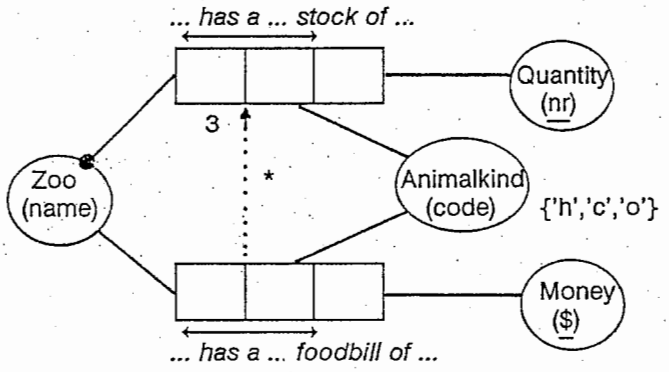


Figure 7.7 An alternative conceptual schema for the Zoo UoD

This schema has an ONF map with two tables:

3 {h,c,o}
Stock (zoo, animalkind, quantity)

Food (zoo, animalkind, foodbill)

But notice the implied subset constraint (this follows from theorem IS3): the set of (Zoo, Animalkind) tuples in the new ternary is a subset of the (Zoo, Animalkind) tuples in the original ternary. This brings into play the *overlap algorithm* which we introduced in NH89 (pp. 233-8) to extend and provide guidelines for the nesting/flattening transformations.

The basic idea is to avoid duplicating the overlap of tuple sets by objectifying the union of the pair types involved. In this case the algorithm dictates that we absorb the subset in the superset, which becomes objectified with two attributes as shown in Figure 7.8.

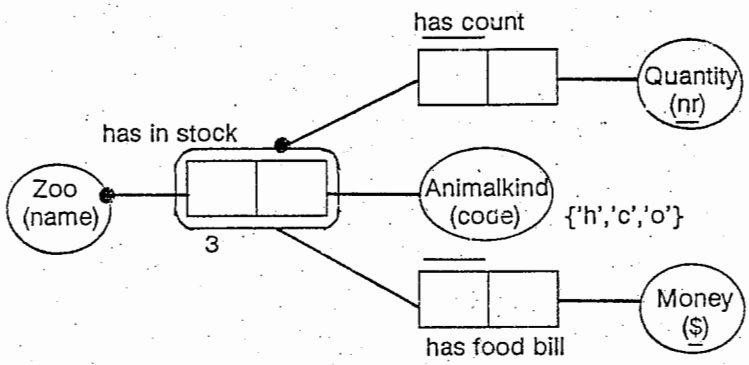


Figure 7.8 An alternative schema based on the overlap algorithm

The overlap algorithm reduces the number of tables in the ONF map. In this case, the schema of Figure 7.8 yields an ONF map with a single table, keyed on the objectified pair type:

3 {h,c,o}
Zoostock (zoo, animalkind, quantity, foodbill OP)

In most cases, this relational schema would be regarded as preferable to the single table option discussed earlier. Note that the role played by Zoo is marked as mandatory in Figure 7.8 since this constraint is to apply even if Zoo plays other roles in the global schema: in such a case this mandatory role constraint must be captured by inter-table equality or subset constraints as discussed earlier.

We now briefly discuss the general overlap algorithm. Though developed independently, our algorithm has some aspects in common with other research on merging relational schemas (Navathe, Sashidhar & Elmasri, 1984). In contrast to the approach of Falkenberg (1988), we permit nesting only when the pair type plays more than one role, or an optional role. In other cases the flattened approach is simpler and preferable, particularly with respect to specification of uniqueness constraints.

Suppose that the schema includes two predicates which include role sequences that are compatible (i.e. the corresponding roles are played by the same primitive object type), and each role sequence is exactly spanned by a uniqueness constraint. Figure 7.9 pictures a simple but common case where each sequence is a contiguous role pair. Let R and S denote the pair types whose populations are the sets of tuples constructed from the objects playing the roles of each sequence. Since R and S are compatible it is meaningful to compare their population for each state.

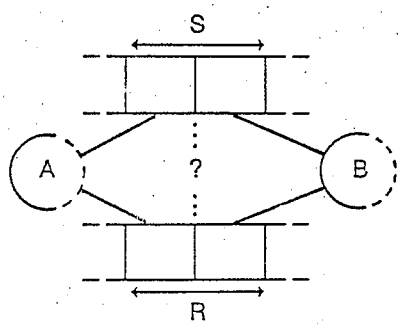


Figure 7.9 The populations of R and S may be compared

Because of the uniqueness constraints, the possibility arises of objectifying (making an explicit pair type from) R and S. In most cases R and S may overlap. While this is allowable, it is generally preferable to avoid duplicating the intersection between R and S by absorbing each into their union. A role sequence is *partial* if its predicate has other roles; otherwise the role sequence is *whole*. The algorithm is summarized below:

Overlap algorithm

case overlap condition **of**

$R \cap S = \{\}$: { *disjoint* }
add an exclusion constraint;

$R \cap S \neq \{\}$ & $R \not\subseteq S$ & $S \not\subseteq R$: { *proper overlap* }
objectify $R \cup S$, attach other roles as a mandatory disjunction;

$R \subseteq S$ & $R \neq S$: { *R is a proper subset of S* }
objectify S, attach other roles of S as mandatory and other roles of R as optional (if R is whole replace it by an attached unary)

$S \subseteq R$ & $S \neq R$: { *S is a proper subset of R* }
objectify R, attach other roles of R as mandatory and other roles of S as optional (if S is whole replace it by an attached unary)

$R = S$: { *identity* }
if R or S is partial then objectify $R \cup S$ and attach other roles as mandatory, else collapse to one relation with rest derived.

end.

In applying this algorithm the designer must choose names for the new predicates, and accept the definitional context specified in our equivalence theorems. An example of the proper subset case was considered earlier (this led to Figure 7.8). As an example of the identity case with partial role pairs, consider a schema comprising the two ternaries: Mine has measured coal reserves of Coalkind in Mass; Mine has indicated coal reserves of Coalkind in Mass. Moreover, let there be an equality constraint between the embedded (Mine,Coalkind) populations for these fact types. We may now transform the schema into a nested version, objectifying the pair type formed from the binary Mine has reserves of Coalkind, with mandatory roles attached for measured and indicated amounts, (and an occurrence frequency of 2 for Mine's role). This maps to a single ONF table whereas the flattened version maps to two tables.

As an example of proper overlap on whole relations, consider the two m:n binaries Person plays Sport and Person coaches Sport, where the populations may properly overlap. This maps to an ONF schema with two tables. Form the binary Person is_involved_in Sport and objectify its pairs, with the following roles attached as a mandatory disjunction: Playing; Coaching. The new schema maps to a single ONF table.

Other examples of this overlap algorithm are included in NH89. We now consider some more complex cases of conceptual schema optimization. Table 7.2 is an output report for a UoD based on an extended version of a classic and awkward problem (see Date 1986, p. 377). Here lecturers are assigned to teach exactly one subject, though many may be assigned to teach the same subject. For each subject taken, students have only one lecturer. Lecturers may teach only the subject which they have been assigned. A lecturer might be assigned a subject but not teach it (e.g. because of poor enrolments).

student	subject	lecturer
Brown A	CS113	Halpin
Brown A	CS102	Rose
Smith J	CS113	Nijssen
Smith J	CS102	Rose
Wang J	CS113	Halpin
?	CS226	Bloggs

Table 7.2 An output report

Let us assume that no lecturer can be a student. Figure 7.10 indicates an attempt to schematize this UoD. For simplicity, reference modes are omitted as they are not germane to our discussion. The uniqueness constraint marked by a broken bar is implied (from IFD2 and IU2), but is shown in case it is not obvious. Prior to our analysis of constraint implication, this schema would have been acceptable in NIAM.

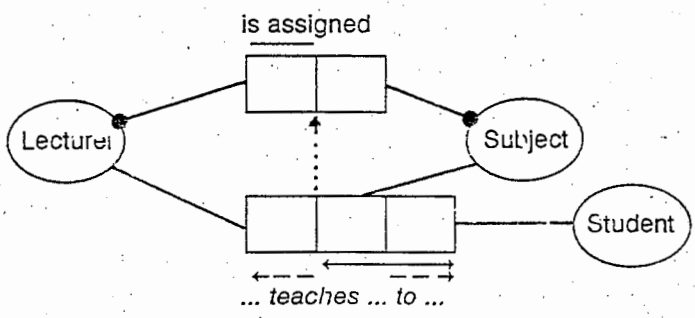


Figure 7.10 A first attempt to schematize Table 7.2

However, from theorem IFD2, the combination of the subset constraint and the uniqueness constraint on the binary implies an FD from the first to the second role of the ternary, i.e. Lecturer functionally determines Subject in this ternary. So the ternary should be split on Lecturer, resulting in Figure 7.11. The equality constraint follows from ESC1, and the lower inter-predicate uniqueness constraint from ESC5. Throughout the transformation process we assume that the old predicates may be retained as derivation rules if desired (the precise form of these rules is given in the equivalence theorems of the previous chapter).

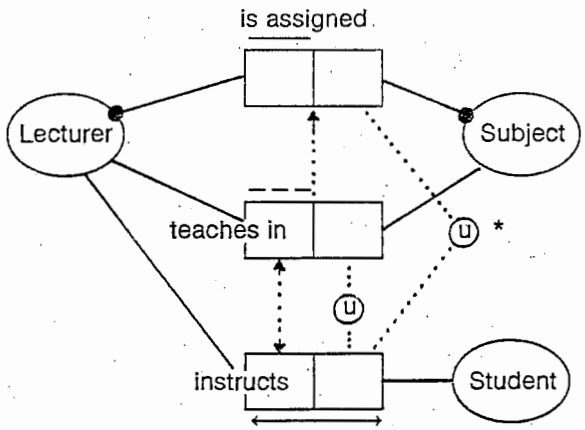


Figure 7.11 Fact types are now elementary

Two implied constraints are shown. The simple UC on the new binary is implied by the UC on the top binary and the subset constraint (theorem IU1); it is also implied by ESC3. The upper inter-predicate UC is implied by the other constraints (theorem IU4). Notice that the keys associated with the subset constraint are now simple. Use of theorem ESS1 now yields Figure 7.12. The formerly implied inter-predicate uniqueness constraint must now be specified since it is no longer implied.

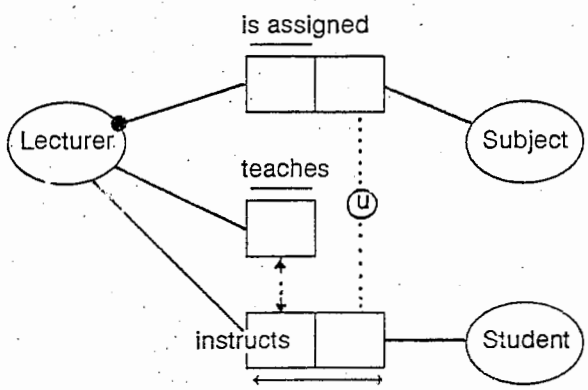


Figure 7.12 Theorem ESS1 has been applied

Now consider the equality constraint. The connection between equality constraints and derivation rules has been discussed in earlier chapters. For implementation purposes it is now convenient to remove the unary from the graphic schema by treating it as a derivation rule; if this rule is no longer required it may simply be dropped. We are now left with the final version (Figure 7.13).

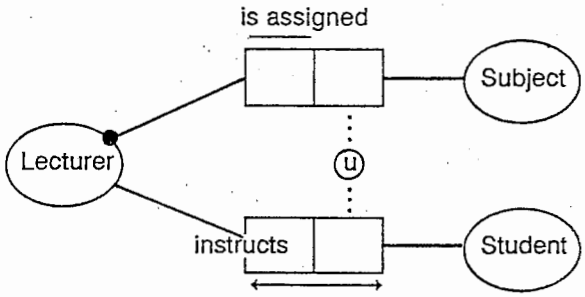
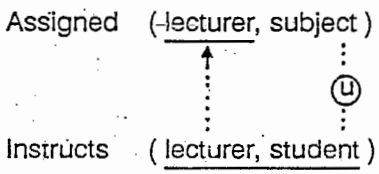


Figure 7.13 The final, optimized version

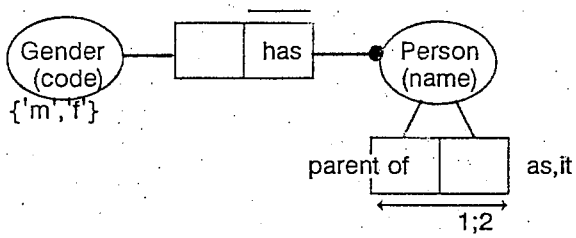
This maps to the following ONF relational schema:



The transformation from the original table structure into this structure is often used as an example of moving from 3rd normal form to Boyce-Codd normal form, at the expense of requiring inter-table constraints. In the previous section we indicated how these constraints may be simply coded in SQL. The main point of our discussion here is to illustrate how such transformations can be visualized diagrammatically and justified from the fact-oriented approach.

It is well known that this particular problem can be neatly handled by functional dependency theory. However one first has to translate the problem into FD theory. Moreover, the process of solving problems in FD theory is often less intuitive and harder to visualize. Minor changes in constraint patterns (e.g. mandatory roles and subset constraints) can significantly increase the complexity of the treatment in FD theory. We believe the approach developed here promotes a high level understanding of such problems as well as highlighting the impact of constraints on the fact types of interest.

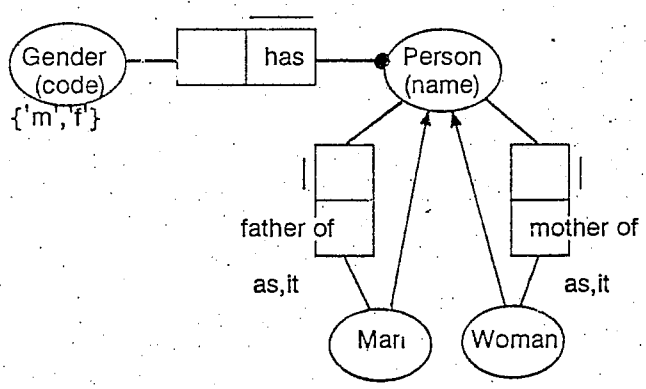
In the previous chapter we discussed several other theorems for replacing composite keys with simple keys. We conclude this section with an example of using a frequency constraint equivalence on a homogeneous fact type. Consider the conceptual schema of Figure 7.14. The textual constraint is added below the diagram since no graphic exists for it: here predicate symbols are abbreviated to one letter.



TC1: { parents of the same child must differ in gender }
$\forall xyzvwu_1u_2(xPz \ \& \ yPz \ \& \ x \neq y \ \& \ xHv \ \& \ vCu_1 \ \& \ yHw \ \& \ wCu_2 \rightarrow u_1 \neq u_2)$

Figure 7.14 A conceptual schema before optimization

Because of the frequency constraint and the capacity to define subtypes in terms of the has_gender predicate, we are able to use theorem EFC1 (actually its reference mode variant) to replace the compositely keyed parent_of predicate with a disjunction of two simply keyed predicates (father_of and mother_of). The result is shown in Figure 7.15 (the definition of parent_of has been omitted).



∀x[Man x ≡ ∃y(x has_gender y & y has_gendercode 'm')]
∀x[Woman x ≡ ∃y(x has_gender y & y has_gendercode 'f')]

Figure 7.15 The optimized version of the previous schema

Unlike the earlier schema, which produces two ONF tables, this schema generates only one table when passed to the ONF algorithm (see Figure 7.16).

The subtype constraints are especially awkward to specify graphically on the relational schema: we use subtype arrows annotated by a restriction condition. Their coding in SQL is however straightforward.

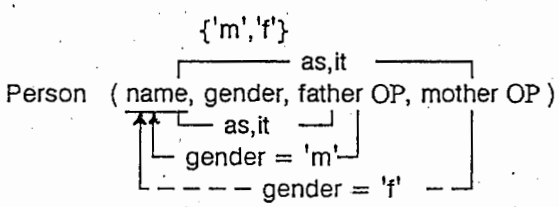


Figure 7.16 The relational schema obtained from Figure 7.15

In this section, optimization of conceptual schemas has been illustrated with some simple examples. For further discussion of related design optimization issues within the relational model, see Amikam (1985) and Diederich & Milton (1988). While the theory developed here can be immediately applied by the schema designer in a freehand form, it is clearly desirable that for large scale schemas such optimization should be automated as much as possible.

We believe that such a conceptual optimizer should be interactive, if only to draw upon the ability of humans to provide meaningful names for new predicates and to check for synonymous object types. Humans also have a role to play in deciding when to consciously strengthen or weaken a schema by adding or deleting definitional context. Ideally, the system should input a conceptual schema and expected query/update pattern, and output an optimized conceptual schema as well as the corresponding relational schema in both shortened and coded form for the desired relational DBMS.

The task of providing automated support for sophisticated conceptual optimization of large global schemas is non-trivial. The next section indicates some of the problems which need to be addressed, and provides some general guidance for selecting transformations on a global schema.

7.3 Optimizing global conceptual schemas

In the previous chapter, several schema equivalence and implication theorems were specified. These allow a conceptual schema to be replaced by another which is contextually equivalent, or at least an acceptable substitute. In the latter case, the substitute schema may be stronger or weaker than the original, but since the theorems identify any differences the resulting information gain or loss is under the conscious control of the designer.

In the previous section, various examples were given of how such theorems may be used to optimize a conceptual schema for implementation in a relational DBMS. However, because there are so many theorems and choices, the designer who wishes to optimize a large global conceptual schema in this way may need some assistance (preferably automated), at least in the way of general guidelines. In this section we briefly indicate some of the relevant issues, and suggest some guidelines for optimizing a global schema. A comprehensive treatment of this topic is beyond the scope of this thesis.

There are three factors which need to be considered when optimizing a conceptual schema: the target system; the query pattern; and the update pattern. The *target system* is the DBMS in which the schema must ultimately be implemented. This might be a relational system (e.g. DB2), an object-oriented system (e.g. Iris), an extended relational system (e.g. PostGres), or even a hierarchic or network database system.

To illustrate the influence of the target system on optimization decisions, recall that in NIAM all conceptual fact types are elementary; in particular, sets are not directly supported as conceptual objects. For example, on a CS we might specify subject enrolments in terms of the *m:n* fact type Student enrolled in Subject, but not as an *n:1* fact type Student enrolled in SubjectSet. Suppose we also record the *n:1* fact type Student born on Date. When mapping to a relational system, because enrolment is *m:n*, enrolment and birthdate facts must go in separate tables. In a system in which set valued fields are supported however, both kinds of fact may go into the same table.

While extending NIAM to support the design of non-relational systems (especially object-oriented systems) is a interesting research topic, in this thesis our treatment of optimization assumes the target system is relational.

The next optimization factor is the *query pattern*. By this we mean the kinds of questions which the system will be expected to answer, together with statistical information about the expected frequency and priority of these questions. Minimally, one needs an estimate of which queries will be issued most often, and which queries (if any) require shorter response times.

Finally, the *update pattern* must be considered, i.e. the kinds, frequencies and priorities of the expected insertions, deletions and modifications to the database tables.

Although space issues are still of some importance, storage technology advances continue to offer dramatic increases in primary and secondary memory at cheaper prices. Hence we focus our optimization efforts largely on reducing the *time* required for responding to the expected query/update pattern, especially the frequent/priority queries/updates (these are sometimes called the "*focussed* transactions").

The continuing trend towards distributed database systems has added considerably to the complexity of design optimization (e.g. by introducing factors such as communication times for message passing between sites, and local versus global query/update patterns). However, for this thesis our optimization guidelines assume we are dealing with a single system.

For a given conceptual schema one can always produce a query/update pattern for which the schema is already optimal. For example, map the CS to its ONF tables. Then let each query take the form "**select * from** *T*", where *T* is one of these tables, and let each update be performed on just one of the tables (although the update aspect might be complicated by join constraints).

With practical applications however, much more complex queries are typically required. The queries which tend to consume the most time are those that involve *joins* or *subqueries*. Usually, it is more important to have rapid response times for queries than for updates. Hence our default optimization strategy aims to *minimize the number of focussed queries which involve joins or subqueries*.

We first consider *joins or subqueries on a single table*. For example, consider the ternary Degree in Year cost Money which records yearly fees for full-time enrolment in degrees. Assume such fees must be recorded for the 3 years 1988-90. This entails that we must know the fees for all three years before we can populate the fact type. Let the corresponding relational table be FTfee(degree, year, fee). Queries such as "How much has the PhD fee increased from 1988 through 1990?" or "What was the average PhD fee for the period 1988-90?" require operations between *different rows* of the base table. For example, the first query may be formulated in SQL using a self-join thus:

```
select  A.fee - B.fee
from    FTfee A, FTfee B
where   A.year = 1990 and A.degree = 'PhD'
  and   B.year = 1988 and B.degree = 'PhD'
```

Notice that the conceptual schema for this ternary fits the pattern of the right-hand alternative in theorem EBT1', as qualified by corollary EBT3. In such cases, if the focussed queries require a join or subquery, we suggest that the ternary by replaced by binaries (see left-hand option of EBT1'). With the present example, this yields the binaries: Degree in 1988 cost Money; Degree in 1989 cost Money; Degree in 1990 cost Money. Let the ONF algorithm group these into the table: FTfee(degree, fee88, fee89, fee90). The queries may now be specified in terms of operations on a single row. For example, the fee increase is obtained from:

```
select  fee90 - fee88
from    FTfee
where   degree = 'PhD'
```

Apart from being easier to formulate, the focussed queries are now faster to execute since the requirement for a join or subquery has been eliminated. This example was trivial and clear-cut. Unfortunately, life is not always so simple. Applying EBT1' to this case requires that a finite enumeration be specified for Year. In our example, Year was restricted to the period 1988-90. First note that in a global schema, Year might play lots of other roles, so this restriction to 1988-90 might not hold globally. However, so long as the years that play in the Fee predicate are restricted to this period (and hence form a subtype FeeYear) we may still transform the ternary into three binaries (though the object type B in the right-hand alternative of EBT1' must now be treated as a subtype .. note that the other enumerated type transformations may be treated similarly).

A more significant issue is the *cardinality of the enumerated type*. Suppose we must record fees for the period 1951..1990. To apply the same transformation theorem in this case would generate 40 binaries, and lead to a relational table with 41 columns: FTfee (degree, fee51, .., fee90). Instead of a very deep table we now have a very wide table. We suggest that once a *specified upper cardinality limit* (5, say) is exceeded, the default guideline for transforming to binaries should be overridden. Certainly a large number of binaries would make the CS awkward to view for humans, and a wide relational table would also be awkward to view (lots of horizontal scrolling) and more difficult to print (except in landscape mode). However, if one can live with such viewing disadvantages, one might still choose to transform to binaries to speed up the focussed queries.

Another related issue is the *stability of the enumeration constraint*. Suppose we need to store the enrolment fees for each year from some specified

start year to the current year. For example, in 1991 we might need to store the fees for 1988-91, in 1992 the fees for 1988-92, and so on. Alternatively, we might be interested in storing fees only for the latest 3-year period. In our formalization of NIAM we ignored all problems associated with schema evolution. But in practice one may need to address such problems.

If all years in the currently designated period must have fees recorded then both the ternary and binary solutions are subject to schema evolution. However, the changes to the ternary solution are less drastic (the enumeration constraint and frequency constraint are changed) than for the binary solution (a new binary fact type is introduced). In terms of the relational tables, the binary solution requires either that an extra column be added each year or that optional columns be added initially in anticipation). If it is not necessary that fees be recorded for all years in a given period, then clearly the ternary solution is stable but the binary solution is not: this advantage may be enough to outweigh concerns about poor performance with focussed queries. These considerations are summarized in the following guideline.

Table Width Guideline (TWG):

> if a subschema matches the form of the binary-ternary equivalence theorem (EBT1 or EBT1'), the enumeration cardinality is small, and focussed queries on the ternary solution require comparisons between different rows, then the binary solution should normally be chosen. In other cases the ternary solution may or may not be preferable.

The previous guideline attempts to reduce the number of focussed queries involving joins or subqueries on the same table. However, the main potential for optimization lies in minimizing the number of focussed queries which require *joins or subqueries on two or more tables*. These kinds of queries tend to be the slowest to execute. To specify a comprehensive minimization algorithm is a major research problem in itself. In this thesis we content ourselves with providing some general guidelines to help reduce the number of such queries. We make no claims as to the completeness of our suggested procedure.

Suppose we have a large global conceptual schema which we wish to optimize in the sense of reducing the number of focussed queries involving multi-table joins or subqueries. Our basic aim is to reshape the schema so that, when passed to the ONF algorithm, it results in fewer tables so that, wherever possible, focussed queries may now take place on single tables rather than multiple tables.

Here are some of the main questions we now need to address:

- Which subschemas are to be selected for optimization?
- For each subschema, which transformations should be applied?
- In what order should these transformations be performed?

In answering these questions we confine ourselves essentially to the schema equivalence and implication theorems discussed earlier in the thesis. To answer the first question, recall that the ONF algorithm maps each fact type with a composite key to a separate relational table (with the usual qualification for compositely defined objects). There are two main situations with the potential for reducing the number of composite keys: (1) *composite keys which are compatible;* (2) *object types with both a simple and a binary key attached.* These situations are summarized in Figure 7.17. In case 1 the composite keys may be binary or longer, but their corresponding object types must be compatible; being elementary, the predicates can have at most one other role not included in the key. In case 2 the predicate with the composite key must be either binary or ternary.

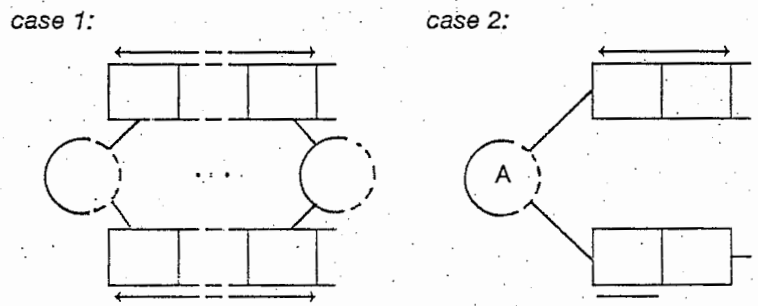


Figure 7.17 Potential patterns for optimization

Hence we select candidate subschemas for optimization by searching the global schema for one of these two patterns. In preparation for this search, we first *flatten* any nested fact types where the pair type plays only one role and this role is mandatory.

If a case 1 pattern is found we immediately apply the *overlap algorithm* to it (see section 7.2). If the pair types overlap at all, then they are replaced by using a single pair type, so the number of composite keys has been reduced. Various examples of the overlap algorithm were cited earlier.

If, in the process of carrying out the optimization, a fact type arises that matches the pattern of one of our splittability theorems then the split designated by the theorem must take place. To illustrate this, as well as the

virtue of preliminary flattening, consider Figure 7.18 (which is based on an example from Bernhard Thalheim).

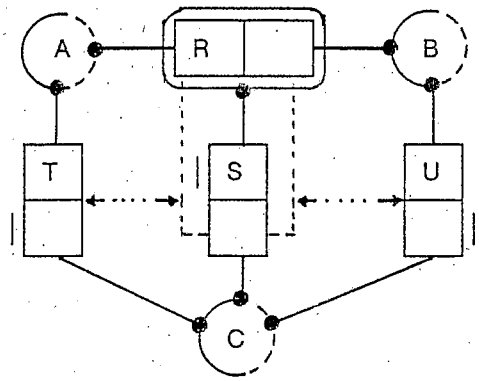


Figure 7.18 The constraints enable substantial optimization

Because the role played by the objectified pair type is mandatory, the nested R-S part should be flattened to a ternary V(a,b,c). Because of the subset constraints and UCs, theorem IFD2 shows that in V we have the FDs: c → b and c → a. So by IFD2 (or IU2) we have a ternary V(a,b,c) which splits on c into V_1(c,a) and V_2(c,b) with an inter-predicate uniqueness constraint between a and b catering for the a,b constraint in V. Because of the equality constraint between V_1 and T, and between V_2 and U, T and U may be specified as derived predicates or simply deleted. This leaves V_1 and V_2 as the only predicates needing to be stored, so the ONF algorithm results in V_1 and V_2 being recombined into V(a,b,c). So whereas before optimization the schema would have generated two tables with inter-table constraints, the optimized schema maps to a single table.

Returning to the optimization procedure for case 1, suppose that flattening and the overlap algorithm have been applied where relevant, but that the *pair types do not overlap* (first subcase of the overlap algorithm). In this case, if the keys span *whole predicates* then, since they are compatible and mutually exclusive, they match the pattern of the left-hand schema in theorem EET4: we should now apply this theorem by introducing a further enumerated object type, transforming to a single, longer fact type. This again has the desired effect of reducing the number of composite keys.

Once case 1 optimizations have been completed, we turn to *case 2*. As it stands, this schema pattern maps to two tables. Basically we search for some way to *convert the binary-keyed fact type into one or two binaries which are simply keyed on A*. If this can be done, the resulting subschema maps to one table keyed on this object type instead of the original two tables. We now summarize ways in which this might be achieved.

If the binary-keyed fact type is a *binary* then let the other object type be B. We list two possibilities for transforming this case. If B has an *enumeration constraint of cardinality 2* then replace it with an object type of cardinality 3 including the "both" option, and replace the predicate with one simply keyed on A (as discussed for Figure 6.20 -- this example may be directly restated as a theorem).

If the compositely keyed binary has a *frequency constraint* of 1;*n* on A's role then, unless *n* is unacceptably large, transform it into *n* exclusive binaries simply keyed on A by using theorem ImFC1 or EFC1

Now consider case 2 where the compositely keyed fact type is a *ternary*. Let the other object type involved in this key be B. We list two possibilities for transforming this case. If B has an *enumeration constraint of cardinality n* then apply the relevant EBT theorem(s) to replace the ternary with *n* binaries simply keyed on A.

If the ternary has a *frequency constraint* of 1;*n* on A's role then, if *n* is small, transform it into *n* binaries simply keyed on A. This particular transformation was not discussed earlier, but all it involves is a simple extension of theorems ImFC1 and EFC1 from the binary to the ternary case.

This overall process is summarized by the *fewer tables procedure* (FTP), so called because it aims to reduce the number of relational tables obtained from the ONF map.

Fewer Tables Procedure (FTP):

1 Flatten any nested fact types where the pair type plays only one role and this role is mandatory.

2 Apply the overlap algorithm to any pattern of compatible composite keys.

3 Use theorem EET4 to transform any case of whole predicates which form compatible, composite but exclusive keys to a single longer predicate.

4 For each case where an object type A has both a simple and binary key attached, where the other role of the binary key is played by B:

```
if the fact type with the binary key is just a binary
    then if B is constrained by {b1,b2} then apply theorem of Fig. 6.20
    else if A's role with B has an FC of 1;n and n is small
            then apply ImFC1 or EFC1
else { the binary-keyed fact type is a ternary }
    if B is constrained by {b1,..,bn}
        then apply relevant EBT theorem(s)
        else if A's role with B has an FC of 1;n and n is small
            then apply ternary version of ImFC1 or EFC1
```

We illustrate the use of FTP on one last example. Consider Figure 7.19, which describes a UoD concerning postgraduate topics offered in the computer science department at the University of Queensland. When passed to the ONF algorithm this schema results in four tables.

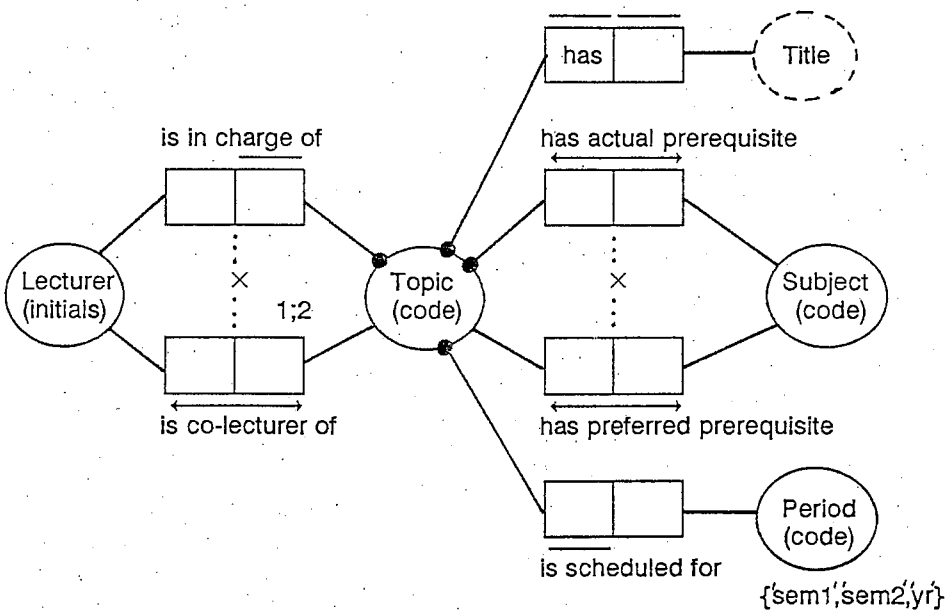


Figure 7.19 This sub-optimal schema generates 4 ONF tables

Suppose a focussed query for this application is: list all topics (code and title) together with their chief-lecturers, co-lecturers, actual pre-requisites and preferred prerequisites. With the ONF schema for Figure 7.19, this query involves a join of 4 tables and hence will be slow to run.

Applying FTP to this schema results in two main changes. Step 3 of the procedure transforms the two prerequisite fact types into the ternary: Topic has Subject as prerequisite with Statuscode {'a','p'}. Note that each topic must have an actual prerequisite: this feature is not captured by theorem EET4 and hence must be added in ternarizing; since no NIAM graphic exists to express it on the ternary we add the abbreviated textual constraint $\forall x[Tx \rightarrow \exists yz(Hxyz \ \& \ z='a')$. This underlines the following two advantages of one advantage of our earlier formalization: we know exactly which features are captured in applying a transformation theorem; additional features may be treated orthogonally.

The second change to the schema results from applying Step 4 of the procedure: this replaces the colecturer binary by two binaries simply keyed on Topic: Lecturer is colecturer1 of Topic; Lecturer is colecturer2 of Topic. The optimized schema results in just two ONF tables, leading to much faster execution of focussed queries.

We recommend that the TWG guideline be used after the FTP procedure has been completed. The optimization strategies presented in this section are by no means comprehensive. We recommend the specification of more detailed optimization strategies as a worthwhile research topic. A workbench to provide automated support for schema optimization is currently in the planning stage. Among other features, such a design tool should provide default optimizations, explain its strategies (including the possible advantages and disadvantages of suggested changes), accept guidance from the designer on information gain or loss as well as better identifiers, and allow the designer to modify its optimization decisions and take a more active part in controlling the transformations.

8 Conclusion

8.1 Summary

This section summarizes what we believe to the main achievements of this thesis. The next section provides a suggested list of related topics for future research.

A prioritized summary of ten contributions was presented in section 1.2. Rather than consider each of these individually, we group them under a few general headings.

The bulk of the thesis has been concerned, in one way or another, with the *formalization of information structures in NIAM*. Building on the foundation of first order predicate logic, we constructed a formal theory general enough to capture all static aspects of NIAM conceptual schemas. To our knowledge, this is the first time that a truly formal framework has been specified in which NIAM theorems may be rigorously stated and proved.

This framework was used to *clarify, refine and extend* many different areas in NIAM. In particular, we believe our axiomatization and semantics for lexical, numeric and described objects has at last provided an adequate account of definite descriptions within the database context, and that our use of contextual definitions to provide inter-translatable conservative extensions of alternative schemas has finally made sense of claims of equivalence and implication in NIAM. Our treatment of derivation rules has clarified the connection between conceptual and implementation concerns, and our orthogonal approach to formalizing schemas, including local/global aspects, has facilitated an incremental, modular approach to schema design.

While we consider this work on theoretical foundations to be the most valuable general contribution in the long term, of the thesis, many *new results* deriving from this work have been presented. These include new theorems on constraint implication, schema equivalence and schema implication, as well as new results on satisfiability and schema formation rules.

With regard to the implementation of conceptual schemas in *relational database systems*, the ONF algorithm has been augmented with constraint mapping, and procedures have been presented for optimizing the conceptual schema prior to executing this algorithm.

We hope to have demonstrated within the thesis that NIAM, in the extended form presented here, provides a knowledge base design methodology that is both intuitively simple and rigorously grounded. Moreover we trust that justification has been provided for the following comment by Dr. E. F. Codd, the founder of the relational model, who questioned a database product designer about support for the existential quantifier, only to receive the reply, "I often get questions of a philosophical nature, but this is the first time I've had a question pertaining to existential philosophy":

> I thought, "what is this data base field doing if the product designers don't know anything about predicate logic?" I feel predicate logic is an essential tool.
> (Codd, quoted in Rapaport 1988)

8.2 Topics for future research

In relation to the subject matter of this thesis, the following research topics are suggested as worthy of future investigation. The suggested implementation modules would complement one another in a comprehensive knowledge base design workbench.

1 Extend the formalization to include dynamic constraints (transition constraints). Temporal logic provides one promising approach to this area.

2 Identify further constraint patterns which are only trivially satisfiable, and compile a more comprehensive list of conceptual schema formation rules (see sec. 6.1)

3 Specify a comprehensive procedure for merging subschemas (sec. 6.1).

4 Develop further constraint implication theorems (sec. 6.2), incorporating relevant existing work from other methodologies (e.g. functional dependency theory).

5 Develop further useful theorems concerning equivalences and implications between conceptual schemas (sec. 6.3).

6 Implement a mapper for translating a conceptual schema into a relational schema in various SQL systems, including complete mapping of constraints (sec. 7.1).

7 Develop further guidelines for optimizing conceptual schemas by transformation prior to the ONF map (secs 7.2-3).

8 Implement a conceptual schema design workbench, capable of supporting conceptual optimization.

9 Implement a computer aided environment for reasoning about conceptual schemas (Appendix II).

10 Explore the connections between NIAM and other approaches, such as object-oriented databases, ER modelling (Appendix III), dependency theory, and distributed databases.

Appendices

Appendix I: The nature and purpose of formalization

As background, this appendix provides a brief overview of the formalization process in general: what it is and why it should be performed. Since our approach is standard, this review is confined to a brief sketch of relevant ideas. Further details may be found in standard works, for example Hunter (1971), Barwise (ed. 1977), and Chang and Keisler (1977).

A language is formal if and only if it can be defined without reference to any interpretation of its formulas (whose well-formedness is decidable). A *formal system* comprises a formal language together with a deductive apparatus (for which the syntactic well-formedness of proofs is decidable). Within the general theory of semiotic (study of signs), formal systems can be treated purely as a matter of syntax (signs as uninterpreted objects) rather than semantics ("meanings" and values of signs) or pragmatics (intended use of signs). Syntax includes the definition of the formal language and proof theory. Before a formal system is applied to a real world problem, appropriate semantics need to be specified; this brings in model theory (the study of interpretations of formal languages).

A *formal language* may be identified with its set of wffs (well formed formulas); it has a vocabulary and a set a formation rules. The *vocabulary* comprises a set of primitive symbols, and optionally a set of defined symbols together with the relevant definitions. The *formation rules* specify how wffs may be constructed from the primitive symbols. Programming and logic provide many examples of formal languages.

Semantics may be formal or informal. An information system deals only with formal semantics: defined symbols have an intension provided by their definitions, but the "meaning" of primitive predicates is given by their possible extensions (relations over formal objects). In contrast, humans may interpret formal objects in terms of the real world and may attach subjective intensional meaning to primitives.

Proof theory is the study of formal systems without reference to interpretation. It deals with the deductive apparatus of formal systems. A deductive apparatus is a set of axioms and/or inference rules. *Axioms* or premises are simply starting formulas rather than "obvious truths" (indeed the notion of truth is outside the scope of proof theory). *Inference rules* (or transformation rules) licence the derivation of some formulas from others.

Proof theory is a purely syntactic game. A *proof* is a finite sequence of formulas each of which is either an axiom or an immediate consequence of earlier formulas by application of the inference rules. A *theorem* is either an axiom, or a formula for which a proof exists. In contrast to our usage, some authors don't count axioms as theorems (e.g. Hughes & Cresswell 1968, p. 16).

An *axiomatic system* has axioms, and usually a very few inference rules. A *natural deduction* or axiomless system has no axioms but does have inference rules (usually several). Axiomatic systems are usually better for proofs of metatheorems (proofs *about* the system). For simple systems (e.g. propositional logic) deduction systems are usually better for proofs *in* the system. For complex systems such as set theory and NIAM knowledge bases, proofs-in are facilitated by axioms and several inference rules.

Model theory or semantics deals with interpretations of formal languages. Its concepts include truth, semantic consequence and logical validity. A *model* of a set of formulas is an interpretation in which every formula in the set is true. A *countermodel* is an interpretation in which at least one formula is false. A formula is a logical truth iff it is true in all interpretations. An argument is a set of propositions one of which (the conclusion) is claimed to follow from the others (the premises). An argument is valid iff in each interpretation in which the premises are true, so is the conclusion.

Metatheory is the theory of formal languages, formal systems and their interpretations. The *object language* is the language that is the object of study. The *metalanguage* is the language used to describe the object language. In Australia English is used as the metalanguage to study a foreign object language, e.g. Japanese. When studying a formal object language, the metalanguage is often English supplemented by logic and set theory notations.

A theorem *in* or of a formal system is a formula with a (syntactic) proof, and has no meaning as such. Informally, a *metatheorem* (theorem *about* a formal system) is a true, meaningful statement about the system, expressed in the metalanguage.

Given a set of formulae in the object language, the metatheory examines such features as consistency, soundness, completeness, decidability and independence. A system S is *simply consistent* iff for no formula A of S are both A and the negation of A theorems of S. A system S is *absolutely consistent* iff some formula of S is not a theorem of S.

A system S is *sound* iff all its theorems are logical truths of S. An inference rule $A_1..A_n/:C$ is sound iff in every interpretation in which $A_1..A_n$ are true so is C. A system S is (weakly) *complete* iff all its logical truths are theorems of S. We ignore other notions of completeness. A system S is *decidable* iff there is an algorithm to decide for each formula of S whether it is a theorem of S. An axiom A of a system S is *independent* of the other axioms iff it cannot be proved from $S - A$ (i.e. S with A removed). A non-independent axiom is *redundant*. Redundancy simplifies proofs in the system but complicates proofs about the system

Two systems are *isomorphic* iff they have the same underlying structure. Theorems in one system can then be mapped onto theorems of the other, e.g. the isomorphism between propositional calculus and set theory enables mapping between $p \vee \sim p \equiv \mathrm{F}$ and $A \cap A' = \{\ \}$. Mapping between formal systems often leads to practical efficiencies. For instance the numeric expression $(42 \times 93)^5$ can be mapped via the logarithm transformation onto the logarithmic system as $(\log 42 + \log 93) \times 5$: this expression can be easily evaluated and the result mapped back via the antilog transformation to provide the solution in the original system.

Mapping between informal and formal systems can also have advantages. Roughly speaking, formal logic and pure mathematics involve the syntactic development of abstract systems, physical science aims to discover isomorphisms between various aspects of physical reality and abstract systems, and applied logic/mathematics pragmatically uses the isomorphisms so discovered. For example, consider Euclidean geometry: formally we may test its axioms for consistency and derive theorems; but whether our space-time continuum is Euclidean is a question for physics, not mathematics.

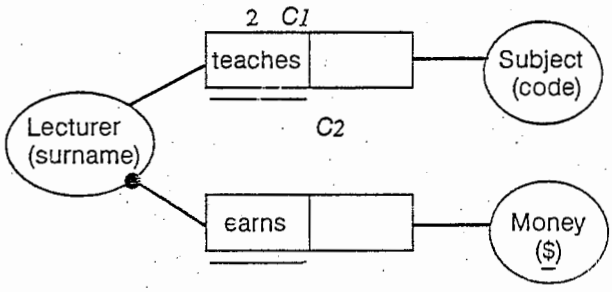
Formalization can help to explain and clarify difficult aspects of the real world, simplify the solution of real world problems, guard against logical errors, and enable new consequences to be deduced. Such formalization is necessary if even some aspects of the original tasks from the real world are to be computerized.

Appendix II: Sample proofs

In this appendix, proofs are by deduction trees. This method combines trees (semantic tableaux) with natural deduction. Background on trees and natural deduction is given in Halpin & Girle (1981). Standard inference rules may be used to make deductions, and thereby shorten the proof and reduce branching. Branching can be entirely eliminated by use of conditional proof if desired. Formulae are ticked (✓) when replaced by equivalent formulae. The justification column on the right annotates the proof; numbers here cite lines used for this step; other abbreviations indicate the formula status, or rules used, e.g.

P	Premise	UI	Universal Instantiation
C	Conclusion	SI	Substitutivity of Identicals
NC	Negated Conclusion	DN	Double Negation
PC	Propositional Calculus tree rule	AA	Affirm the Antecedent (i.e. Modus Ponens)
QN	Quantifier Negation		
EI	Existential Instantiation	AB	Affirm one side of a Biconditional

Theorem: The following CS (Figure 6.1, p. 6-2) is not strongly satisfiable

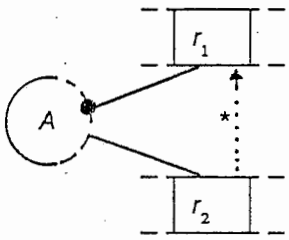


Proof 1:

This follows since the Teaches predicate cannot be consistently populated, as shown below. We abbreviate this predicate as T, and translate only the relevant part of the CS.

✓	1	∃xy xTy	Populate T
	2	∀xyz(xTy & xTz → y = z)	C2
	3	∀xy[xTy → ∃z(xTz & y ≠ z)]	C1 conjunct
	4	aTb	1 EI
	5	aTb → ∃z(aTz & b ≠ z)	3 UI
✓	6	∃z(aTz & b ≠ z)	4,5 AA
✓	7	aTc & b ≠ c	6 EI
	8	aTc	7 PC
	9	b ≠ c	7 PC
	10	aTb & aTc → b = c	2 UI
	11	aTb & aTc	4,8 conj
	12	b = c	10,11 AA
		×	9,12

Theorem: If A plays r_1 and r_2, and r_1 is mandatory then a subset constraint from r_2 to r_1 is implied (p. 6-9, theorem IS1), i.e.



Strictly speaking, there is a result of this form for each choice of A, r_1 and r_2. The proof-scheme may be expressed in agonizing, low level detail by selecting arbitrary predicates R and S (not necessarily distinct) of arities n and m, arbitrary role positions i and j, and expanding "x plays r_1" and "x plays r_2" as "x ϵ R.i" and "x ϵ S.j", which expand as "$\exists x_1..x_n(Rx_1..x_n$ & x = x_i)" and "$\exists x_1..x_m(Rx_1..x_m$ & x = x_m)". However, since x is free in these latter formulae, and our human insight reveals that the internal structure of these formulae is irrelevant to the proof, we symbolize membership in the role populations simply as:

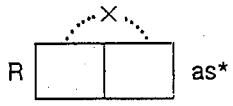
P_1x = x plays r_1
P_2x = x plays r_2

We symbolize only the relevant aspects of the given CS in premises 1 and 2, and show that denying the subset constraint leads to a contradiction. The argument is just a simple AAA syllogism (i.e. subsethood is transitive).

Proof 2:

	1	$\forall x(P_2x \rightarrow Ax)$	P { from TPN (typing constraint) }
	2	$\forall x(Ax \rightarrow P_1x)$	P { TMR2 (r_1 is mandatory for A) }
✓	3	$\sim\forall x(P_2x \rightarrow P_1x)$	NC { negate subset constraint }
✓	4	$\exists x\sim(P_2x \rightarrow P_1x)$	3 QN
✓	5	$\sim(P_2a \rightarrow P_1a)$	4 EI
	6	P_2a	5 PC
	7	$\sim P_1a$	5 PC
	8	$P_2a \rightarrow Aa$	1 UI
	9	Aa	8,6 AA
	10	$Aa \rightarrow P_1a$	2 UI
	11	P_1a	10,9 AA
		×	7,11

Theorem: If R is a binary with mutually exclusive roles then R is asymmetric (p. 6-13), i.e.

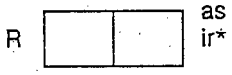


Proof 3:

The exclusion constraint is translated by TXC1 as $\forall x \sim(x \in R.1 \ \& \ x \in R.2)$, and the $\in$ notation is then replaced (see sec. 4.2) to yield the premise. This is our first example with branching (see step 8).

	1	$\forall x \sim(\exists y\ xRy \ \& \ \exists y\ yRx)$		P { df × }
✓	2	$\sim\forall xy(xRy \rightarrow \sim yRx)$		NC { ~ df as }
✓	3	$\exists xy \sim(xRy \rightarrow \sim yRx)$		2 QN
✓	4	$\sim(aRb \rightarrow \sim bRa)$		3 EI
	5	aRb		4 PC
	6	bRa		4 PC,DN
✓	7	$\sim(\exists y\ aRy \ \& \ \exists y\ yRa)$		1 UI
✓	8	$\sim\exists y\ aRy$	$\sim\exists y\ yRa$	7 PC
	9	$\forall y \sim aRy$	$\forall y \sim yRa$	8 QN
	10	$\sim aRb$	$\sim bRa$	9 UI
		×	×	5,10; 6,10

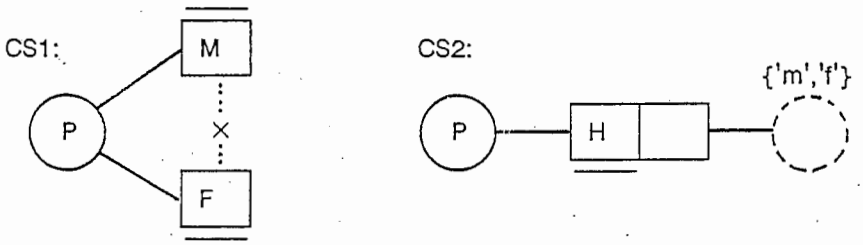
Theorem: If R is an asymmetric binary then R is irreflexive (p. 6-12).



Proof 4:

	1	$\forall xy(xRy \rightarrow \sim yRx)$	P { df as }
✓	2	$\sim\forall x \sim xRx$	NC { ~ df ir }
✓	3	$\exists x\ xRx$	2 QN,DN
	4	aRa	3 EI
	5	$aRa \rightarrow \sim aRa$	1 UI
	6	$\sim aRa$	5,4 AA
		×	4,6

Theorem: The subschemas CS1 and CS2, when extended by D1 and D2 respectively, are logically equivalent (p. 6-29, Fig. 6.18).



D1: ∀xy(xHy ≡ Mx & y='m' ∨ Fx & y='f')

D2: ∀x(Mx ≡ xH'm')
∀x(Fx ≡ xH'f')

Proof 5:

The subschemas translate into KL as the following specific axioms, together with the KS axioms. From theorem CC≠, we know that 'm' ≠ 'f'.

CS1: ∀x(Px → Described x)
∀x(Mx → Px)
∀x(Fx → Px)
∀x~(Mx & Fx)

CS2: ∀x(Px → Described x)
∀xy[xHy → Px & (y='m' ∨ y='f')]
∀xyz(xHy & xHz → y=z)

We prove the result by showing each representation implies the other (the common axiom that P is described is safely omitted). A *proof window* may be opened at any time to consider a subproblem (e.g. the proof of one conjunct in the global conclusion). For a more general windowing technique for interactive proof editors, see Robinson & Staples (1988).

Notice the way branching is reduced by resolving truth values. For a set of such resolution rules see Halpin & Girle (1981, pp. 141-2): each such rule may be specified as a derived rule of inference.

For efficiency reasons, we have modified the branching rule for implication to agree with the approach of Staples. So a node of the form $\alpha \rightarrow$ ß branches to $\sim\alpha$; α,ß rather than to $\sim\alpha$; ß.

The last window of the full proof includes an example of the modified implication branching rule (Part 2 of proof:/:C4, lines 11,12 right branch): this avoids having to prove closure for ~aHb on the right branch.

Part 1: CS1 & D1 ⇒ CS2 & D2

1	∀x(Mx → Px)	P	
2	∀x(Fx → Px)	P	
3	∀x~(Mx & Fx)	P	
4	∀xy(xHy ≡ Mx & y='m' ∨ Fx & y='f')	P	
/:	∀xy[xHy → Px & (y='m' ∨ y='f')]	/:	C1
	∀xyz(xHy & xHz → y=z)		C2
	∀x(Mx ≡ xH'm')		C3
	∀x(Fx ≡ xH'f')		C4

/: C1

✓ 5	~∀xy[xHy → Px & (y='m' ∨ y='f')]				NC1
✓ 6	∃xy~[xHy → Px & (y='m' ∨ y='f')]				5 QN
✓ 7	~[aHb → Pa & (b='m' ∨ b='f')]				6 EI
8	aHb				7 PC
✓ 9	~[Pa & (b='m' ∨ b='f')]				7 PC
✓ 10	aHb ≡ Ma & b='m' ∨ Fa & b='f'				4 UI
✓ 11	Ma & b='m' ∨ Fa & b='f'				8,10 AB
12	Ma → Pa				1 UI
13	Fa → Pa				2 UI
14	~Pa		✓ ~(b='m' ∨ b='f')		9 PC
15			b≠'m'		14b PC
16			b≠'f'		14b PC
17	Ma	Fa	Ma	Fa	11 PC
18	b='m'	b='f'	b='m'	b='f'	11 PC
19	Pa	Pa	×	×	12,17;13,17;15,18;16,18
	×	×			14,19

/: C2

✓ 5	~∀xyz(xHy & xHz → y=z)				NC2
✓ 6	∃xyz~(xHy & xHz → y=z)				5 QN
✓ 7	~(aHb & aHc → b=c)				6 EI
8	aHb				7 PC
9	aHc				7 PC
10	b≠c				7 PC
✓ 11	aHb ≡ Ma & b='m' ∨ Fa & b='f'				4 UI
✓ 12	aHc ≡ Ma & c='m' ∨ Fa & c='f'				4 UI
✓ 13	Ma & b='m' ∨ Fa & b='f'				8,11 AB
✓ 14	Ma & c='m' ∨ Fa & c='f'				9,12 AB
✓ 15	~(Ma & Fa)				3 UI
16	Ma		Fa		13 PC
17	b='m'		b='f'		13 PC
18	~Fa		~Ma		15,16
19	Ma	Fa	Ma	Fa	14 PC
20	c='m'	c='f'	c='m'	c='f'	14 PC
21	b=c	×	×	b=c	17,20;18,19;18,19;17,20
	×			×	10,21

		/: C3
✓	5 ~∀x(Mx ≡ xH'm')	NC3
✓	6 ∃x~(Mx ≡ xH'm')	5 QN
	7 ~(Ma ≡ aH'm')	6 EI
✓	8 aH'm' ≡ Ma & 'm'='m' ∨ Fa & 'm'='f'	4 UI
✓	9 aH'm' ≡ Ma & T ∨ Fa & F	8 R=,CC≠
✓	10 aH'm' ≡ Ma ∨ F	9 Id,CF
✓	11 aH'm' ≡ Ma	10 Id
	12 Ma ≡ aH'm'	11 Com
	×	7,12

		/: C4
✓	5 ~∀x(Fx ≡ xH'f')	NC4
✓	6 ∃x~(Fx ≡ xH'f')	5 QN
	7 ~(Fa ≡ aH'f')	6 EI
✓	8 aH'f' ≡ Ma & 'f'='m' ∨ Fa & 'f'='f'	4 UI
✓	9 aH'f' ≡ Ma & F ∨ Fa & T	8 CC≠,R=
✓	10 aH'f' ≡ F ∨ Fa	9 CF,Id
✓	11 aH'f' ≡ Fa	10 Id
	12 Fa ≡ aH'f'	11 Com
	×	7,12

□

Part 2: CS2 & D2 ⇒ CS1 & D1

1	∀xy[xHy → Px & (y='m' ∨ y='f')]	P	
2	∀xyz(xHy & xHz → y=z)	P	
3	∀x(Mx ≡ xH'm')	P	
4	∀x(Fx ≡ xH'f')	P	
/:	∀x(Mx → Px)	/:	C1
	∀x(Fx → Px)		C2
	∀x~(Mx & Fx)		C3
	∀xy(xHy ≡ Mx & y='m' ∨ Fx & y='f')		C4

		/: C1
✓	5 ~∀x(Mx → Px)	NC1
✓	6 ∃x~(Mx → Px)	5 QN
✓	7 ~(Ma → Pa)	6 EI
	8 Ma	7 PC
	9 ~Pa	7 PC
✓	10 Ma ≡ aH'm'	3 UI
	11 aH'm'	8,10 AB
✓	12 aH'm' → Pa & ('m'='m' ∨ 'm'='f')	1 UI
✓	13 aH'm' → Pa & (T ∨ F)	9 R=,CC≠
✓	14 aH'm' → Pa & T	10 Id
	15 aH'm' → Pa	11 Id
	16 Pa	11,15 AA
	×	9,16

/: C2

✓ 5	~∀x(Fx → Px)	NC2
✓ 6	∃x~(Fx → Px)	5 QN
✓ 7	~(Fa → Pa)	6 EI
8	Fa	7 PC
9	~Pa	7 PC
✓ 10	Fa ≡ aH'f'	4 UI
11	aH'f'	8,10 AB
✓ 12	aH'f' → Pa & ('f'='m' ∨ 'f'='f')	1 UI
✓ 13	aH'f' → Pa & (F ∨ T)	9 CC≠,R=
✓ 14	aH'f' → Pa & T	10 Id
15	aH'f' → Pa	11 Id
16	Pa	11,15 AA
	×	9,16

/: C3

✓ 5	~∀x~(Mx & Fx)	NC3
✓ 6	∃x(Mx & Fx)	5 QN,DN
✓ 7	Ma & Fa	6 EI
8	Ma	7 PC
9	Fa	7 PC
✓ 10	Ma ≡ aH'm'	3 UI
✓ 11	Fa ≡ aH'f'	4 UI
12	aH'm'	8,10 AB
13	aH'f'	9,11 AB
14	aH'm' & aH'f' → 'm'='f'	2 UI
15	aH'm' & aH'f'	12,13 Conj
16	'm'='f'	14,15 AA
	×	16,CC≠

/: C4

✓ 5	~∀xy(xHy ≡ Mx & y='m' ∨ Fx & y='f')			NC4
✓ 6	∃xy~(xHy ≡ Mx & y='m' ∨ Fx & y='f')			5 QN
✓ 7	~(aHb ≡ Ma & b='m' ∨ Fa & b='f')			6 EI
8	Ma ≡ aH'm'			3 UI
9	Fa ≡ aH'f'			4 UI
✓ 10	aHb → Pa & (b='m' ∨ b='f')			1 UI
11	~aHb		aHb	10 PC
12			✓ Pa & (b='m' ∨ b='f')	10 PC
13	✓Ma & b='m' ∨ Fa & b='f'		Pa	7,11; 12
14			✓ b='m' ∨ b='f'	12
15	Ma	Fa	✓~(Ma & b='m' ∨ Fa & b='f')	13; 7,11
16	b='m'	b='f'	~(Ma & b='m')	13; 15
17	~aH'm'	~aH'f'	~(Fa & b='f')	11,16; 15
18	aH'm'	aH'f'		8,9,15
19	×	×	b='m' / b='f'	17,18; 14
20			aH'm' / aH'f'	11,19
21			Ma / Fa	8,9,20
22			Ma & b='m' / Fa & b='f'	21,19
			× / ×	16,17,22

□

Appendix III: Entity-Relationship modelling

In this thesis we have formalized and extended the information state component of the fact-oriented modelling approach known as NIAM (Nijssen's Information Analysis Method). In particular, equivalence and implication between conceptual schemas has been rigorously examined, and guidelines have been given for optimizing a conceptual schema by applying transformation theorems before mapping it to a relational schema. Despite the strengths of NIAM, in current practice the most popular data modelling formalism is *Entity-Relationship modelling* (ER). Some of the benefits of NIAM over ER have been cited elsewhere (e.g. Nijssen, Duke & Twine 1988). In this appendix we briefly compare NIAM with ER, with a view to making our results more accessible to users of ER.

ER was first proposed by Chen (1976). Basically, ER views the real world as consisting of *entities* which have *attributes* (properties, e.g. gender) and participate in *relationships*. The domain of discourse also includes *values*. A value is basically a typed constant (e.g. 'Codd':surname, '5':integer, 'male':gender); at the implementation level, value typing is typically implicit. An entity is a thing with independent existence, and a relationship is an association amongst entities.

In Chen's original proposal, relationships may have attributes but do not participate in relationships. Chen proposed a diagram notation in which entity types and binary relationship types were depicted as named boxes and diamonds respectively. Line segments connecting boxes to diamonds were labelled "1", "n" or "m" to indicate whether the relationship type was *1:1*, *1:n*, *n:1* or *m:n*. Attributes and value sets were not shown on the diagram, but were listed separately.

Chen's original version of ER is grossly lacking in expressibility. It cannot specify mandatory roles, frequency constraints, subtyping, exclusion constraints, subset constraints, equality constraints, some uniqueness constraint patterns (e.g. overlapping and intra-predicate UCs), and so on. Since most of our formal results rely on the capacity to express such features, most of our conceptual schema transformations cannot be rigorously mapped into original ER. Of course, informally the ER designer may use some of the basic ideas, but there is no formal way of controlling information loss with this approach.

Many researchers, including Chen, have since extended ER to enable such features to be captured. The resulting models are usually called EER (Extended ER). Some of these models come very close to the expressibility of NIAM. Unfortunately, there are dozens of different EER models. These differ not only in the features that they support but in the diagram notation used to picture such features.

For example, the relationship diamond might be replaced with polygons (depending on the arity of the predicate) or simply deleted in favour of adding role names to parts of the relationship line. Attributes might be included as names attached by lines to their entity types; sometimes attribute names are enclosed in ellipses. A 1:n fact type might be specified with a "crow's foot" symbol at one end. Mandatory and optional roles might be distinguished by solid and broken lines, or by double and single lines, or by shading a "role" within the relationship polygon. Moreover, the same symbol may be used with different meanings depending on the EER model (e.g. a double line might be used to denote a multi-valued attribute instead of a mandatory role).

Of the many EER notations with which we are familiar, none appears to be as simple or intuitive as the NIAM notation. In general, the more complex the constraint situation, the more awkward the EER diagram becomes in comparison with the NIAM diagram. For example, the typical EER notations make it difficult if not impossible to graphically specify cases such as disjunctive mandatory roles (inclusive and exclusive), subset constraints between compatible pair types, and complex subtype graphs. Part of the problem with most EER notations is their failure to focus on the roles in a relationship type. This not only makes it hard to express constraints on roles but also makes it extremely awkward to populate schema diagrams with fact instances for validation purposes.

Because there is no standard EER notation, and EER notations are typically awkward, we later suggest a version of our own which is closer to NIAM. So long as the conceptual features are supported in some way, the designer should be able to translate his or her particular EER notation into ours. If the concept is not supported in the designer's EER graphical notation, then the concept will have to be specified textually (e.g. by a logical formula).

Let us suppose that we have an EER notation that is as expressive as the NIAM notation. To use the theorems and optimization guidelines developed in this thesis, one needs to provide an algorithm for translating between the two

notations. It is beyond the scope of this thesis to specify such an algorithm in detail. However, we do sketch the essential aspects.

Once one has a sufficiently expressive EER model, it is clear that there is more similarity than difference between NIAM and EER. Entity types, value sets, and relationship types of EER basically correspond to entity types, label types and relationship types in NIAM. Our specialized treatment of definite descriptions, string types and number types can be easily applied to EER. There is only one fundamental difference in the conceptual basis of the two formalisms: the notion of *attributes*. The attribute concept is explicitly emphasized in EER, but is not required in NIAM (though it is definable in terms of NIAM's primitives). We now illustrate how attributes in EER can be translated into the NIAM framework.

The EER notion of an attribute may be summarized as follows. An attribute is *simple* if it cannot be decomposed into simpler attributes, else it is *composite*. For example, Gender is a simple attribute but Birthdate is composite if we wish to individually access Birthyear, Birthmonth, and Birthday. An attribute is *single-valued* if it maps each member of its entity type onto at most one value (e.g. Gender is a *function* from Person to the *value set* {'male':gender, 'female':gender}). A value set is also called a *domain*. A *multi-valued* attribute maps each member of its entity type onto at most one *set* of values. For example, LanguagesSpoken is a function from Person to the *power set* of languages (where each language is treated as a value, e.g. 'Japanese':language).

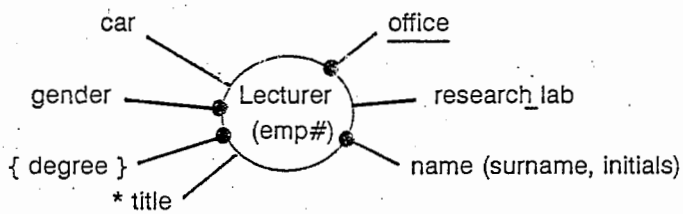
Each entity type must have an attribute (simple or composite) as its primary *identifier*. Each other attribute is a *descriptor*. If an attribute can be derived from others by means of a derivation rule it is said to be *derived* (e.g. assuming all grades are recorded, GradeAverage may be derived from Grade). Different attributes may be defined over the same domain. For example, Birthdate and Enrolmentdate may be defined over the domain Date; Salary and Tax may be defined over the domain Money.

For non-attribute aspects of our EER diagram notation we use the NIAM notation. For example, entity types are shown as named ellipses (not boxes) and relationship types are shown as named box-sequences (not diamonds), and the NIAM constraint notations are used.

We now specify a graphic notation for displaying attributes on a diagram. Each attribute has a (local) name. However, the local names "name", "code" and "nr" may be used for different attributes, so the entity type name is prepended to the local attribute name to provide a qualified attribute name.

Attributes which are primary identifiers are shown as NIAM reference modes, i.e. in parentheses below the entity type name. Unlike our NIAM diagrams, we do not underline numerically based reference modes. The domains of attributes are typically listed in a separate table: for domains that are subsets of String or Real, we use the NIAM notation. We chose numbers instead of strings for employee numbers (e.g. we might want to specify simple rules for generating new employee numbers).

Descriptors have their (local) names written at the end of a line segment connected to their entity ellipse. If a descriptor is a 1:1 attribute, it is underlined. If a descriptor is mandatory, a mandatory role dot is added to the point where it is connected to it entity type. Some other constraints (e.g. exclusion constraints between roles played by the entity type) may be added. If a descriptor is derived it is preceded by an asterisk, and the derivation rule is written separately. Composite attributes have their components listed in parentheses, and multi-valued attributes are delimited by braces. See Figure 1.




```
* if 'PhD' in degree(Lecturer)
      then title(Lecturer) = 'Dr'
      else if gender(Lecturer) = 'f'
            then title(Lecturer) = 'Ms'
            else title(Lecturer) = 'Mr'
```

attribute	*domain*
emp#	[dddd]
car	<dddaaa>:car
gender	{'m','f'}:gender
office	<d4>:room
research_lab	<d4>:room
surname	<a20>
initials	<a3>
degree	<a7>:degree

Figure 1 Some attributes of Lecturer

The translation of Figure 1 into NIAM is shown in Figure 2. As an alternative, an object type LecturerName could have been introduced, identified by surname and initials. The main thing to note is that with single-valued attributes the role played by Lecturer has a simple uniqueness constraint, but this is not true for the multivalued attribute. Note that the predicate names can be generated by prepending "has" to the attribute names (recall our shortened predicate name rule for "has").

The object type names at the far-end of the predicate may be assigned the domain names. Of course, the designer would often override the generated predicate names with better ones (e.g. "was awarded" instead of "has_degree"). In this example we have used meaningful names for the reference modes (e.g. 'reg#'): these need to be supplied by the designer (if desired, default names could be automatically generated by appending "_id" to the domain name, e.g. "car_id").

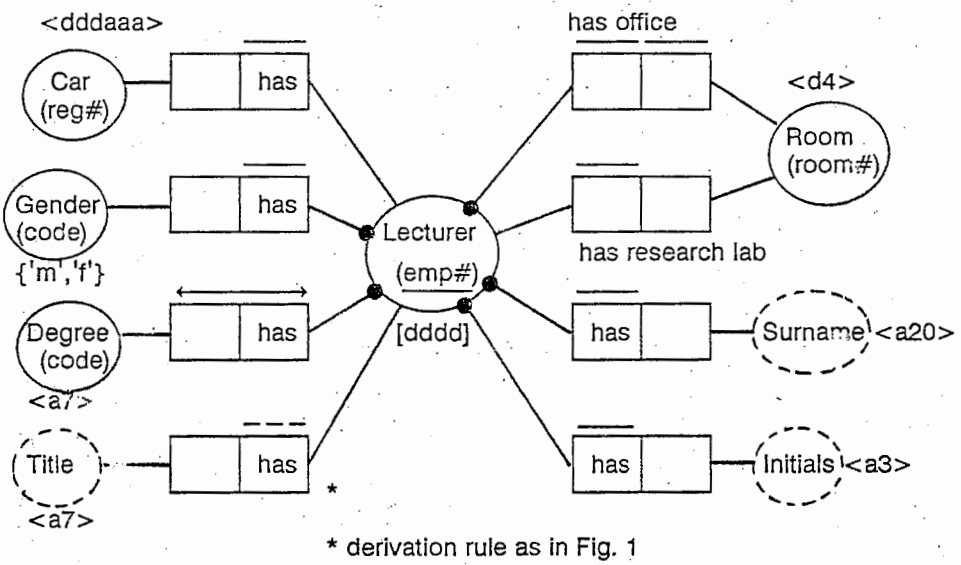


Figure 2 Figure 1 translated into NIAM

A quick glance at Figures 1 and 2 indicates that if one wishes to view the attributes of Lecturer without the domain details, then the EER diagram at the top of Figure 1 might be preferred for this purpose since it is more compact. Indeed, it is sometimes argued that NIAM's notation is less suitable than EER's because a NIAM diagram packs all the information onto one figure, swamping the human viewer with too much detail.

There is no doubt that in working with conceptual schemas there is a need to support *information hiding* in various ways. An automated design tool should enable the designer to choose how much of the conceptual schema he or she wishes to view at a particular time. If it is desired to view just a single entity type and its attributes without domain detail, then we have no objection to a view such as that at the top of Figure 1 being used: this could be accepted within NIAM simply as an incomplete abbreviation of the relevant fact types. Of course, other kinds of information hiding should also be supported. For example, the designer may wish to choose which fact types to view (e.g. show a local subset of the entity types and their fact types, excluding "attributes"), which kinds of constraint to see, and so on.

Having said this, we emphasize that there are many occasions in which it is extremely advantageous to be able to see all the relevant constraints at once. In such situations NIAM diagrams can make it easier to spot important constraints, and to suggest further optimization. In contrast, unless managed very carefully by the designer, EER diagrams can be dangerous because they hide relevant information. For example, consider Figure 3 (this is a simpler version of a problem discussed in section 7.3).

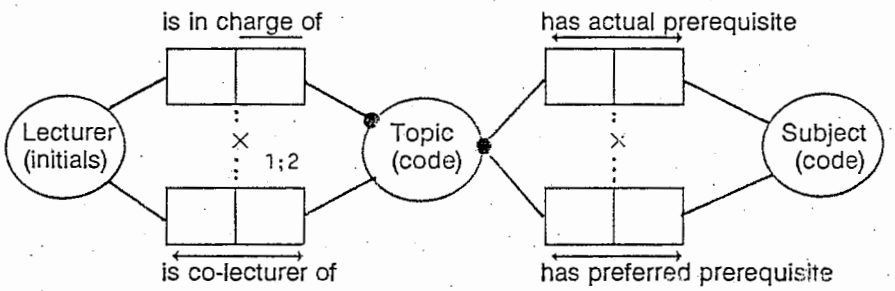


Figure 3 A NIAM schema in need of optimization

Using EER, a designer might come up with a schema diagram equivalent to that shown in Figure 4.

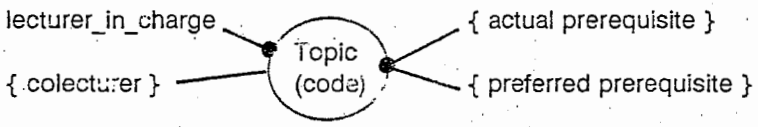


Figure 4 An EER schema missing important constraints

Clearly, Figure 4 does not capture the following constraints which are captured in Figure 3: there are at most two colecturers for a given topic; a lecturer in charge of a topic cannot be a colecturer of the same topic; a subject which is an actual prerequisite for a topic cannot be a preferred prerequisite for that topic. Apart from being important constraints to enforce, the frequency constraint and the right-hand exclusion constraint enable the NIAM schema to be optimized so that only two tables (instead of four) are generated when the schema is passed to the ONF algorithm (see section 7.3).

The moral here for EER is to avoid use of multi-valued attributes when a frequency constraint is involved, or when there is more than one such attribute based on the same domain. With this guideline, one can produce an EER diagram for this example using four relationship types; and if the NIAM

notation for predicates and constraints is adopted we get the same diagram as the earlier NIAM diagram. This can now be optimized in the same way.

It can be seen from this example (and many others) that one problem with EER is that important connections might go unnoticed by not revealing the domains of attributes on the diagram. Another related problem with EER is that there are too many choices as to how to capture an aspect of the UoD. A classic example here is the 1:1 head-of-department fact type: is Head an attribute of Department, Department an attribute of Lecturer, or do we have a relationship type instead?

Moreover, in EER we may need to revise our schema when some aspect chosen as an attribute (e.g. { subjects } as an attribute of Student) must be recast as a relationship (e.g. we may later decide to record the credit point value of each subject). NIAM avoids the need for such agonizing choices and revision, by proving a simple, direct and uniform treatment of all fact types.

In summary, EER-style diagrams can be useful for providing external views which hide unwanted detail; but in order to use such diagrams safely one needs to be sure that the hidden detail is actually irrelevant to the task at hand. Just what aspects are relevant might not always be obvious. If one wishes to deal rigorously with the notions of schema equivalence and implication then information about domains and constraints needs to be made explicit. NIAM provides a convenient way of displaying this on a single diagram.

Bibliography

During preparation of this thesis, the extensive literature on relational database design has been kept under review; but works of this nature which are not directly relevant to this thesis have not been cited here.

Amikam, A. 1985, 'On the Automatic Generation of Optimal Internal Schemata', *Inform. Systems*, vol. 10, no. 1, Pergamon, pp. 37-45.

Andrews, J. 1987, *Trilogy User's Manual*, Complete Logic Systems, Vancouver, Canada.

Asimov, I. 1986, *Foundation and Earth*, Collins, London.

Australian Government Publishing Service 1988, *Style Manual for Authors, Editors and Printers*, 4th ed., AGPS Press, Canberra.

Barwise, J. (ed.) 1977, *Handbook of Mathematical Logic*, North-Holland Publishing Co., Amsterdam.

Bledsoe, W.W. & Loveland, D.W. (eds) 1984, *Automated Theorem Proving: After 25 Years*, American Mathematical Society, Providence.

Beeri, C. 1980, 'On the Membership Problem for Functional and Multivalued Dependencies in Relational Databases', *ACM TODS*, vol. 5, no. 3, pp. 241-259.

Beeri, C. & Kifer, M. 1986, 'An Integrated Approach to Logical Design of Relational Database Schemes', *ACM TODS*, vol. 11, no. 2, pp. 134-158.

Brachman, R.J. 1988, 'The Basics of Knowledge Representation and Reasoning', *AT&T Tech. Journal*, vol. 67, no. 1, pp. 7-24.

Bradley, R & Swartz, N. 1979, *Possible Worlds*, Basil Blackwell, Oxford.

Bry, F. & Manthey, R. 1986, 'Checking Consistency of Database Constraints: a Logical Basis', *Proc. Twelfth International Conf. on Very Large Data Bases*, VLDB, Kyoto, pp. 13-20.

Bubenko, J.A. jr 1986, 'Information System Methodologies - A Research View', *Information Systems Design Methodologies: Improving the Practice*, eds T.W. Olle, H.G. Sol & A.A. Verrijn-Stuart, North-Holland, Amsterdam.

Casanova, M.A. & Tucherman, L. 1988, 'Enforcing Inclusion Dependencies and Referential Integrity', *Proc. of the Fourteenth Conf. on Very Large Data Bases*, VLDB, Los Angeles, pp. 38-48.

Chamberlin, D.D., Astrahan, M.M., Eswaran, K.P., Griffiths, P.P., Lorie, R.A., Mehl, J.W., Reisner, P. & Wade, B.W. 1976, 'SEQUEL 2: A Unified Approach

to Data Definition, Manipulation and Control', *IBM J. Res. Develop.*, Nov. 1976, pp. 560-75.

Chang, C.C. & Keisler, H.J. 1977, *Model Theory*, 2nd edn, North-Holland Publishing Co., Amsterdam.

Chen, P.P. 1976, 'The Entity-Relationship Model - Toward a Unified View of Data', *ACM Transactions on Database Systems*, vol. 1, no. 1, pp. 9-36.

Chen, P.P. & Dogac, A. 1983, 'Entity-Relationship Model in the ANSI/SPARC Framework', *Entity-Relationship Approach to Information Modelling and Analysis*, ed. P.P. Chen, Elsevier Science Publishers B.V., North Holland.

Codd, E.F. 1970, 'A Relational Model of Data for Large Shared Data Banks', *Communications of the ACM*, vol. 13, no. 6, pp. 377-87.

Date, C.J. 1986a, *An Introduction to Database Systems*, vol. 1, 4th edn, Addison-Wesley, Reading MA.

Date, C.J. 1986b, *Relational Database: Selected Writings*, Addison-Wesley, Reading MA. pp. 269-310.

Date, C.J. 1987, *A Guide to The SQL Standard*, Addison-Wesley, Reading MA.

D'Atri, A. & Sacca, D. 1984, 'Equivalence and Mapping of Database Schemes', *Proceedings of the 10th International Conference on Very Large Data Bases*, VLDB, Singapore, pp. 187-95.

Debenham, J.K. 1985, 'Knowledge Base Design', *The Australian Computer Journal*, vol. 17, no. 1, pp. 42-8.

Diederich, J. & Milton, J. 1988, 'New Methods and Fast Algorithms for Database Normalization', *ACM Transactions on Database Systems*, vol. 13, no. 3, pp. 339-365.

Falkenberg, E.D. 1976, 'Concepts for Information Modelling', *Modelling in Data Base Management Systems*, ed. G.M. Nijssen, North-Holland, Amsterdam.

Falkenberg, E.D. 1986, *Data Bases and Information Systems I: Lecture Notes*, University of Nijmegen, The Netherlands.

Falkenberg, E.D. 1988, 'Deterministic Entity Relationship Modelling', discussion paper for FRISCO workshop.

Gallaire, H., Minker, J. & Nicolas, J.-M. 1984, 'Logic and Databases: A Deductive Approach', *ACM Computing Surveys*, vol. 16, no. 2.

Girle, R.A., Halpin, T.A., Miller, C.M. & Williams, G.H. 1978, *Inductive and Practical Reasoning*, Rotecoge, Brisbane, pp. 149-57

Gottlob, G. & Zicari, R. 1988, 'Closed World Databases Opened Through Null Values', *Proc. of the Fourteenth Conf. on Very Large Data Bases*, VLDB, Los Angeles, pp. 50-61.

Haack, S. 1978, *Philosophy of Logics*, Cambridge University Press, London.

Halpin, T.A. & Girle, R.A. 1981, *Deductive Logic*, 2nd edn, Logiqpress, Brisbane.

Halpin, T.A. 1986a, 'Conceptual Schemata and Predicate Logic', *Proceedings of the First Australian AI Congress*, Melbourne.

Halpin, T.A. 1986b, 'Logic Diagrams and Query Formulation in SQL', *Technical Report 77*, Dept of Computer Science, University of Qld.

Halpin, T.A. 1987, 'Information Design and IPT', *Computers in the Curriculum: Proc. of 1987 CEGQ Conference*, Computer Educ. Group of Qld, Brisbane.

Halpin, T.A. 1988a, *CS112 Lecture Notes*, Dept of Computer Science, University of Queensland, Brisbane.

Halpin, T.A. 1988b, 'Information Systems Design', *Technology and Autonomy: Proc. of 1988 CEGQ Conference*, Computer Educ. Group of Qld, Brisbane, pp. 16-21.

Halpin, T.A. 1988c, 'Conceptual Schema Transformations', Dept of Computer Science, University of Queensland, Brisbane.

Halpin, T.A. 1988d, *CS113 Lecture Notes*, Dept of Computer Science, University of Queensland, Brisbane.

Halpin, T.A. 1989, 'Venn Diagrams and SQL Queries', *The Australian Computer Journal*, vol. 21, no. 1, pp. 27-32.

Hughes, G.E. & Cresswell, M.J. 1968, *An Introduction to Modal Logic*, Methuen, London.

Hunter, G. 1971, *Metalogic: An Introduction to the Metatheory of standard First-Order Logic*, Macmillan, London.

ISO 1982, *Concepts and Terminology for the Conceptual Schema and the Information Base*, ed. J.J. van Griethuysen, ISO TC97/SC5/WG3, Eindhoven.

Jardine, D.A. & Reuber, A.R. 1984, 'Information Semantics and the Conceptual Schema', *Information Systems*, vol. 9, no. 2, pp. 147-56.

Jardine, D.A. & van Griethuysen, J.J. 1987, 'A logic-based information modelling language', *Data and Knowledge Engineering*, vol. 2, North-Holland, pp. 59-81.

Kent, W. 1978, *Data and Reality*, North-Holland, Amsterdam.

Kent, W. 1982, 'Choices in Practical Data Design', *Proc. of the Eigth Int. Conf. on Very Large Data Bases*, VLDB, pp. 165-180.

Kent, W. 1986, 'The Realities of Data: Basic Properties of Data Reconsidered', *Database Semantics*, eds T.B. Steel Jr & R.A. Meersman, Elseviers Science Publishers B.V., North Holland.

Kobayashi, I. 1984, 'Validating Database Updates', *Inform. Systems*, vol. 9, no. 1, Pergamon Press, pp. 1-17.

Kobayashi, I. 1986, 'Losslessness and semantic correctness of database schema transformation: another look at schema equivalence', *Inform. Systems*, vol. 11, no. 1, Pergamon Press, pp. 41-59

Leung, C.M.R. & Nijssen, G.M. 1987, 'From a NIAM Conceptual Schema into the Optimal SQL Relational Database Schema', *Australian Computer Journal*, vol. 19, no. 2, pp. 69-75.

Leung, C.M.R. 1988, 'On Design and Implementation of a Fifth Generation Information System', PhD thesis, Dept of Computer Science, University of Queensland.

Levesque, H.J. 1984, 'A Fundamental Tradeoff in Knowledge Representation and Reasoning', *Proc. CSCSI-84*, London, Ontario, 1984, pp. 141-152

Lindsay, P.A. 1988, 'A survey of mechanical support for formal reasoning', *Software Engineering Journal*, Jan. 1988, pp. 3-27.

Loux, M.J. (ed.) 1979, *The Possible and the Actual*, Cornell University Press. Ithaca.

Lundberg, B. 1983, 'On Correctness of Information Models', *Inform. Systems*, vol. 8. no. 2, Pergamon Press, pp. 87-93.

Luk, W.S. & Kloster, S. 1986, 'ELFS: English language From SQL', *ACM Transactions on Database Systems*, vol. 11, no. 4, pp. 447-472.

Maier, D. 1983, *The Theory of Relational Databases*, Computer Science Press, Potomac, Md.

Mark, L. 1987, 'The Binary Relationship Model - 10th Anniversary', *Proc. VIM-47 EMDA Conf.*, Minneapolis, Nov. 1987.

McGrath, G.M. 1987, 'The Transition to Fifth Generation technology: Conceptual Schema Implementation', *The Australian Computer Journal*, vol. 19, no. 1, Feb. 1987.

Meersman, R.A. 1981, 'RIDL: A Query System as Support for Information Analysis', *ECODO*, vol. 32, September 1981.

Meersman, R.A. 1988, 'The Future of Relational Database Design', *Proc. Oracle User Conf.*, Paris.

Meyer, J., Weigand, H. & Wieringa, R. 1988, 'Specifying Dynamic and Deontic Integrity Constraints', Rapport IR-175, Vrije Universiteit Amsterdam.

Morgenstern, M. 1984, 'Constraint Equations: Declarative Expression of Constraints with Automatic Enforcement', *Proc. of the Tenth Int. Conf. on Very Large Data Bases*, VLDB, Singapore, pp. 291-300.

Motro, A. 1986, 'Completeness Information and Its Application to Query Processing', *Proc. of the Twelfth Int. Conf. on Very Large Data Bases*, VLDB, Kyoto, pp. 170-178.

Navathe, S.B., Sashidhar, T. & Elmasri, R. 1984, 'Relationship Merging in Schema Integration', *Proc. of the Tenth Int. Conf. on Very Large Data Bases*, Singapore, pp. 78-90.

NH89: see entry for Nijssen & Halpin 1989.

Nijssen, G.M. & Falkenberg, E.D. 1983, *Design of Conceptual Schemata and Data Bases*, Lecture notes, Dept. of Computer Science, University of Queensland (106 pp.).

Nijssen, G.M. 1985, 'On Experience with Large Scale Teaching and Use of Fact-based Conceptual Schemas in Industry and University', *Database Semantics: Proc. IFIP Conf. on Database Semantics*, eds R. Meersman & T. Steel, North Holland Publishing Co., Amsterdam.

Nijssen, G.M., Duke, D.J. and Twine, S.M. (1988), The Entity-Relationship Model Considered Harmful, *Proc. 6th Symposium on Empirical Foundations of Information and Software Sciences*, Atlanta, Georgia.

Nijssen, G.M., Gillner, R., Halpin, T.A., Mansfield, T., Sargent, M. & Willmore, R. 1988, 'The General Architecture for Information Systems - Illustrative example: Inventory Control System - (Predicate Logic)', Working paper for the IFIP WG 8.1 Task Group FRISCO, Dept of Computer Science, University of Queensland.

Nijssen, G.M. & Halpin, T.A. 1989, *Conceptual Schema and Relational Database Design*, Prentice Hall, Sydney.

Olle, T.W., Hagelstein, J., Macdonald, I.G., Rolland, C., Sol, H.G., Van Assche, F.J.M. & Verrijn-Stuart, A.A. 1988, *Information Systems Methodologies - A Framework for Understanding*, Addison-Wesley, Wokingham, England.

Osborn, S.L. & Heaven, T.E. 1986, 'The Design of a Relational Database System with Abstract Data Types for Domains', *ACM Transactions on Database Systems*, vol. 11, no. 3, pp. 357-73.

Rapaport, M. 1988, 'Dr E.F. Codd: The relational model and beyond', *Database Programming and Design*, Feb. 1988.

Reeves, S. 1987, 'Semantic Tableaux as a framework for Automated Theorem Proving', *AISB87*, Edinburgh.

Reiter, R. 1984, 'Towards a Logical Reconstruction of Relational Database Theory', *On Conceptual Modelling*, eds M.L. Brodie, J. Mylopoulas & J.W. Schmidt, Springer-Verlag, New York, pp. 191-233.

Rennie, M.K. & Girle, R.A. 1973, *Logic: Theory and Practice*, University of Queensland Press, Brisbane.

Robinson, P. & Staples, J. 1988, 'A Logic Architecture for Computer Aided Reasoning', *Austral. Comp. Sci.Communications*, vol. 10, pp. 215-224.

Rybinski, H. 1987, 'On First-Order Logic Databases', *ACM Transactions on Database Systems*, vol. 12, no. 3, pp. 325-49.

Sadri, F. 1987, 'Multi-relation Dependencies', *Inform. Systems*, vol. 12, no. 2, Pergamon, pp. 145-9.

Shoval, P. 1985, 'Essential Information Structure Diagrams and Database Schema Design', *Inform. Systems*, vol. 10, no. 4, Pergamon, pp. 417-423.

Sowa, J.F. 1988, 'Knowledge Representation in Database, Expert Systems, and Natural Language', *Proc. IFIP WG2.6/WG2.8 Working Conf. on the Role of AI in Database and Inf. Systems*, eds C.H. Kung & R.A. Meersman, North Holland, Amsterdam.

Tarski. A. 1949, 'Arithmetical classes and types of Boolean algebras' (Preliminary report), *Bull. Am. Math. Soc.*, vol. 55, no. 64.

Teorey, T.J., Yang, D. & Fry, J.P. 1986, 'A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model', *ACM Computing Surveys*, vol 18, no. 2, pp. 197-222.

Verheijen, G.M.A. & Van Bekkum, J. 1982, 'NIAM: An Information Analysis Method', *Information Systems Design Methodologies: a comparative review*, eds T.W. Olle, H.G. Sol & A.A. Verrijn-Stuart, IFIP, North Holland, Amsterdam.

Vermeir, D. 1983, 'Semantic Hierarchies and Abstractions in Conceptual Schemata', *Inform. Systems*, vol. 8, no. 2, Pergamon Press, pp. 117-24.

Vermeir, D. & Nijssen, G.M. 1982, 'A Procedure to Define the Object Type Structure of a Conceptual Schema', *Inform. Systems*, vol. 7, no. 4, pp. 329-336.

Zaniolo, C. & Melkanoff, M.A. 1982, 'A Formal Approach to the Definition and the Design of Conceptual Schemata for Database Systems', *ACM Transactions on Database Systems*, vol. 7, no. 1, pp. 24-59.

Index of main acronymns and theorems

main acronyms and abbreviations:

constraint implication theorems:

Conceptual schema equivalence and implication theorems

name	*page*	*category*
EBT1..3	6-36	binary-ternary conversion
EET1..5	6-38..42	enumerated type
EFC1..3	6-47,8	frequency constraint
ENF1..3	6-48,9	nest/flatten
EOA1,2	6-49	object type addition/deletion
ERP	6-39	role permutation
ESC1..8	6-43..5	split/combine
ESS1	6-51	subset constraint
EUB1	6-35	unary-binary conversion
-	6-32	unclassified equivalence (see Fig. 6.20)
ImFC1..3	6-46	implied schema based on FC
ImOA1	6-50	implied schema based on object type deletion