CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

A GRAPHICAL SYSTEM FOR LOGICAL RELATIONAL DATABASE DESIGN (GLORD)

by

Kassem Afif Saleh

Thesis

submitted to the School of Graduate Studies

in partial fulfillment of the

requirements for the degree of

Master of Science

in

Computer Science

ISBN 0-315-33245-X

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

To my wife

and my parents

ACKNOWLEDGMENTS

I would like to thank Dr. To-Yat Cheung for his valuable time, patience and advice throughout my graduate work. I am also grateful to many friends and colleagues who supported me during my research. Thanks to the University of Ottawa and NSERC for the financial support during the early stages of my research.

TABLE OF CONTENTS

List of Figures

List of Tables Page

ABSTRACT

A graphical model based on an extension of Petri-nets is developed and implemented as an interactive system for investigating the following logical database design problems: i) Given a set of data attributes and a set of data constraints described in terms of data dependencies, logically group these attributes into relation schemes such that they satisfy the various normal forms. ii) Given a database scheme which satisfies a given set of data constraints, test whether or not the scheme satisfies other constraints. Like an ordinary Petri-net, the model has places and transitions. Each place represents an attribute of the universe of a database scheme. There are three kinds of transitions, namely FD-, MVD- and R-transitions, representing functional dependencies, multivalued dependencies and relation schemes, respectively. A subset of the R-transitions may represent a join dependency. In comparison with other existing models, our model has the following advantages: i) It can handle the above-mentioned three kinds of transitions. ii) It is based on a strong mathematical foundation for handling FD-transitions and R-transitions. iii) It has unique representations for three types of data dependencies. iv) It is clear and appealing to

viewers' intuition. v) It is easy for computerization. The interactive graphical system consists of three subsystems: i) The graphic support subsystem which provides the user with an interactive interface. ii) The design and test subsystem which is used to test implications, to find closure and to generate third, fourth and Boyce-Codd normal forms. iii) Help subsystem which provides explanations to the user. The graphic system is being expanded to include a subsystem used as a graphical query language for database manipulation. The main contribution of the thesis is in the area of graphical programming rather than in the algorithmic aspects of relational database design.

Chapter I

INTRODUCTION

1.1 PROBLEMS AND MOTIVATION OF THE THESIS

In this thesis, we develop a Petri-net-based model and implement an interactive graphical system for investigating the following logical database design problems: i) Given a set of data attributes and a set of data constraints described as data dependencies, logically design the database scheme (i.e., logically group these attributes into relation schemes) such that the relation schemes satisfy the various normal forms with respect to these constraints. ii) Given a database scheme which satisfies a given set of data constraints, test whether or not the scheme satisfies other constraints.

Investigation of these problems requires the solution of some membership or implication-related problems and the generation of normalized relation schemes by decomposition or synthesis.

Recent developments in the following three areas of database research are the motivations behind the research works of this thesis:

i) Graphical representations of data constraints and relation schemes.

ii) Automation of design processes.

iii) Graphical systems and languages for database manipulation and design.

Althqugh many graphical methods (see Section 2.3.1 for a review) are available, most of them either do not provide unique representations or are not suitable for computerization. An ideal graphical model should be able to uniquely represent the objects, appeal to the viewers' intuition, have regular shapes for the convenience of implementation, and has a strong mathematical foundation for the development of solutions to application problems and derivation of complex properties. For example, Derivation directed acyclic graphs (DDAG) [MAIE83] have been used for representing functional dependencies (FD). However, it does not always uniquely represent a set of FDs. Hypergraphs [MAIE83], for example, have been used for the study of database schemes for many years. However, they are difficult and compliaated to view and draw for ordinary users and very inconvenient for automation because of their irregular shapes. Recently, Cheung [CHEU85] has proposed a Petri-net-based model for representing three different types of data constraints and relational database schemes. The model has unique representations for data constraints and powerful mathematical foundation based on Petri-nets and

hypergraphs. It is appealing to the user's intuition and convenient for automation.

Logical design of relational databases has gained a strong theoretical foundation during the last decade. Although there exist a lot of theoretical results and algorithms, we know of only two computerized systems for automating these processes [BJOR83, CERI86]. Both systems are not graphical and are implemented in the logic programming language PROLOG. In this thesis, we only consider the first one implemented by Bjornstedt. As far as we know, our system is the first graphical package serving such purposes.

The graphical approach is also gaining momentum in the research of programming languages, though not much has been done in the areas of database manipulation and design [MATW85, TAMA83, ZHAN83]. Based on our model, a programming language for manipulating databases is being developed [CHEU86].

1.2 SUMMARY OF THE THESIS

This thesis modifies, implements and expands the applications of a graphical model proposed by Cheung [CHEU86] for the logical design of relational database schemes. (For details, see Section 3.3). Its main contribution is in the area of graphical programming rather

than in the algorithmic aspects of relational database design. This model appeals visually to the users, has unique representations, is easy for implementation, and has a powerful mathematical foundation. The model is an extension of Petri-nets. Like an ordinary Petri-net, it has places and transitions. Each place represents an attribute of the universe of a database scheme. There are three kinds of transitions: (i) FD-transitions, denoted by single bars, are the same as the transitions of an ordinary Petri-net. They are used to represent functional dependencies. (ii) MVD-transitions, denoted by double bars, represent multivalued dependencies. (iii) R-transitions, denoted by squares, represent relation schemes. The undirected edges emanating from an R-transition are connected to those attributes of the scheme it represents. Each figure containing a set of regular-shaped R-transitions is equivalent to an irregular-shaped hypergraph, widely-used for database studies. However, the former is much easier for observation and drawing and hence for computerization. A subset or the entirety of the R-transitions may be used to represent a join dependency defined on the relevant relation schemes.

This model has been implemented on a Macintosh computer as a user-friendly graphic system called GLORD. Graphically, the user can create and modify database schemes and investigate several logical design problems with respect to

a given set of data constraints: To find the closure and dependency basis of a set of attributes, to test implications, and to generate third, fourth and Boyce-Codd normal forms. At the end, the designer has a minimal set of data dependencies and a set of relation schemes that logically satisfy the design objectives.

The modular design of the system software allows future addition of more types of data dependencies and inclusion of a graphical language for database manipulation and other Petri-net-based application systems, such as systems for protocol synthesis, etc.

In the rest of this chapter, we first introduce the different types of data dependencies and normal forms. We then briefly describe two common approaches for relational database design. In Chapter 2, we present a survey on three approaches for expressing and testing the design procedures and a description of the existing tools for such purposes. In Chapter 3, an extension to Cheung's Petri-net-based model [CHEU86] is introduced and compared with some of the existing graphical models. Chapters 4 and 5 give detailed descriptions of GLORD from a user's point of view and from a system's point of view, respectively. In Chapter 6, some examples and concluding remarks are given.

1.3 FUNDAMENTALS OF LOGICAL DESIGN OF RELATIONAL DATABASES

This section introduces two classes of constraints, namely data dependencies and normal forms.

A relation scheme R is a finite set of attributes R = { A1, A2, A3,.....}. A set of relation schemes forms a database scheme D. The set of all attributes forms the universe of attributes U of the database scheme. The domain of possible values of an attribute Ai is denoted by Di.

A tuple of a relation r is a mapping that associates a value with each attribute in R. If S is a subset of R and t is a tuple in r, then t(S) denotes the tuple obtained by restricting the mapping to S. The projection of r onto S, denoted by r(S), is the set of all tuples t(S) with duplications omitted, where t is in r.

A relation instance r of R is a finite set of tuples { t1 ,t2 ,t3 ; .... } with the restrictions that each tj $\in$ r, tj(Ai) $\in$ Di. Hereafter, we shall omit the word instance and simply say 'a relation' if it is clear from the context.

1.3.1 Data Dependencies

In the following, r always represents a relation instance of a relation scheme R.

Semantic constraints governing the relationships among the values of the attributes of each tuple of r are called data dependencies. Among the many types of data dependencies, this thesis is concerned with functional, multivalued and join dependencies. They will be explained later.

Other types of data dependencies have also been investigated in the literature. A list follows.

- Embedded multivalued dependency (EMVD) studied by both Fagin [FAGI77] and Delobel [DELO78].
- Embedded join dependency (EJD) studied by Fagin [FAGI77] and Delobel [DELO78].
- Implicational dependency (IMD) studied by Fagin [FAGI80].
- Algebraic dependency (AD) studied by Yannakakis and Papadimitriou [YANN80].
- Generalized dependency (GD) studied by Beeri and Vardi [BEER81] and Sadri and Ullman [SADR80].
- Template dependency (TD) studied by Sadri and Ullman [SADR82].
- Inclusion dependency (ID) studied by Casanova, Fagin and Papadimitriou [CASA84].

1.3.1.1 Functional Dependencies (FD)

This type of data dependencies is the most common one.

Definition 1.1. Let X and Y be subsets of R. We say that r satisfies the functional dependency (FD) X --> Y if for any two tuples t1 and t2 in r, t1(Y) = t2(Y) whenever t1(X) = t2(X).

Definition 1.2. Given a set F of FDs, an FD X --> Y is said to be logically implied by F, denoted as F |= X --> Y, if every relation that satisfies the FDs in F also satisfies X --> Y.

The following well-known Armstrong's inference axioms for FDs [ARMS74] are sound and complete for FDs, meaning that they lead to valid FDs only. They can be used to derive other valid FDs.

- Axiom 1. Reflexivity: X --> X always holds for any r.
- Axiom 2. Augmentation: If r satisfies X --> Y and Z is a subset of R, then r satisfies XZ --> Y.
- Axiom 3. Transitivity: If r satisfies both X --> Y and Y --> Z, then r satisfies X --> Z.

1.3.1.2 Multivalued Dependencies (MVD)

This type of data dependencies was introduced by Fagin [FAGI76] and Zaniolo [ZANI76] independently.

Definition 1.3. Let X and Y be two disjoint subsets of R and Z = R - (XY). A relation r obeys the MVD X ->-> Y if for any two tuples t1 and t2 of r such that t1(X) = t2(X), there exists a tuple t3 of r satisfying t3(X) = t1(X), t3(Y) = t1(Y) and t3(Z) = t2(Z). We say that X multidetermines Y.

The following is a set of inference axioms for multivalued dependencies [MAIE83]:

- Axiom 1. Reflexivity: Relation r satisfies X ->-> X.

- Axiom 2. Augmentation: If r satisfies X ->-> Y, then r satisfies XZ ->-> Y.

- Axiom 3. Additivity: If r satisfies X ->-> Y and X ->-> Z, then r satisfies X ->-> YZ.

- Axiom 4. Projectivity: If r satisfies X ->-> Y and X ->-> Z, then r satisfies X ->-> Y Z and X ->-> Y - Z.

- Axiom 5. Transitivity: If r satisfies X ->-> Y and Y ->-> Z, then r satisfies X ->-> Z - Y.

- Axiom 6. Pseudotransitivity: If r satisfies X ->-> Y and YW ->-> Z, then r satisfies XW ->-> Z- YW.

- Axiom 7. Complementation: If r satisfies X ->-> Y and Z = R- XY, then r satisfies X ->-> Z.

Two other inference axioms are available when F contains both FDs and MVDs.

• Axiom 1. Replication: If r satisfies X --> Y, then r satisfies X ->-> Y.

• Axiom 2. Coalescence: If r satisfies X ->-> Y and Z -> W, where $W \subseteq Y$ and $Y \cap Z = \emptyset$, then r satisfies X -> W.

Definition 1.4. The closure of a set of attributes X, denoted by X+, with respect to a given set of FDs, is the set of all attributes of U such that X functionally determines any subset of X+.

Definition 1.5. Let F be a set of FDs and MVDs. The dependency basis of a set of attributes X, denoted by DEP(X), is a partition of U into pairwise disjoint subsets Y1, Y2,... such that an MVD X ->-> Y is implied by F if and only if Y-X is a union of some of the Yi's.

An FD (X --> Y) or MVD (X ->-> Y) is logically implied by F if Y is a union of some elements in DEP(X). [BEER80, GALI82, HAGI79, MAIE81].

1.3.1.3 Join Dependencies (JD)

This type of data dependencies is a generalization of multivalued dependencies. In the following, we introduce the concepts of natural joins, lossless join decomposition and join dependency.

Definition 1.6. The natural join of a set of relations r1, r2, ... rn of relation schemes R1, R2, ... Rn, respectively, denoted by r1 ⋈ r2 ⋈ ... ⋈ rn, is the relation satisfying all tuples t over the attribute set R1 ∪ R2 ∪ ... ∪ Rn, such that t(Ri) is in ri for each i.

Definition 1.7. Let R1, R2, ... Rk be subsets of R. We say that a relation r satisfies the join dependency JD: *[R1,R2,.....,Rk] if r = R1(r) ⋈ R2(r) .... ⋈ Rk(r).

Definition 1.8. Let R be decomposed into subschemes R1, R2, R3... Rk and F be a set of data dependencies. We say that the decomposition is a lossless join decomposition (with respect to F) if for every relation r of R satisfying F satisfies the join dependency *[R1,R2, ... Rk].

It can be shown that r satisfies a MVD X ->-> Y if it satisfies *[XY,XZ], where Z = R - X Y.

1.3.2 Normal Forms (NF)

A normal form is some combination of data dependencies satisfied by a special subset of attributes (e.g., keys) of a relation scheme.

Several normal forms have been introduced in the literature. Codd originally defined the 1NF, 2NF and 3NF. [CODD72]. All normalized relations are in 1NF. Some of the 1NF relations are in 2NF and some 2NF relations are in 3NF. A revised version of the 3NF [CODD75] is called BCNF. It is more restrictive than the 3NF. Fagin [FAGI77] also defined 4NF.

A database scheme D is said to be in 2NF, 3NF or 4NF if each of its relation schemes is in 2NF, 3NF or 4NF, respectively.

1.3.2.1 First Normal Forms (1NF)

R is in 1NF if the values of every attribute A in R are atomic. That is, these values are not lists or sets of values or composite.

1.3.2.2 Second Normal Forms (2NF)

In order to define 2NF we have to introduce the following concepts.

A key K is a subset of R having the following property: if ti ,tj $\in$ r then ti(K) $\neq$ tj(K), and no proper subset K' of K has this property. As there may be more than one key for a relation, a key can be designated as the primary key. The term candidate key is used to denote any minimal set of

attributes that can be a key. A superkey is any superset of a key. An attribute is said to be non-key or non-prime if it does not participate in any key.

Definition 1.9. Y is said to be fully dependent on X if it is functionally dependent on X and not functionally dependent on any proper subset of X. The FD X --> Y is said to be left-reduced if Y is fully dependent on X. Y is called a determinant if some other set of attributes is fully dependent on it.

Definition 1.10. R is in 2NF if and only if it is in 1NF and every non-key attribute is fully dependent on the primary key.

1.3.2.3 Third Normal Forms (3NF)

Definition 1.11. Given a subset X of R, an attribute A in R and a set F of FDs, A is said to be transitively dependent upon X in R if there is a subset Y of R with X --> Y, Y $\not\rightarrow$ X and Y --> A under F and A $\notin$ X $\cup$ Y.

Definition 1.12. R is in 3NF with respect to a set of FDs if it is in 1NF and no non-prime attribute in R is transitively dependent upon a key of R.

Generally, a designer should at least aim at 3NF relation schemes as the goal of a design.

1.3.2.4 Boyce-Codd Normal Forms (BCNF)

This new restriction on relation schemes is due to Boyce and Codd and is hence called BCNF [CODD74]. It is conceptually simpler than 3NF.

Definition 1.13. A set of attributes is called a determinant if some other set of attributes is fully functionally dependent on it.

Definition 1.14. R is in BCNF if and only if every determinant is a candidate key.

Note that BCNF considers not only the primary key but also all possible keys for a relation scheme.

1.3.2.5 Fourth Normal Forms (4NF)

Definition 1.15. R is in 4NF if there exists MVD X ->-> Y, where Y is not empty or a subset of X and X U Y does not include all the attributes of R, such that X is a superkey of R.

It has been proven that any relation scheme can be losslessly decomposed into an equivalent collection of 4NF relation schemes. If R is in 4NF with respect to a set F of FDs and MVDs, then R is in BCNF with respect to the set of FDs implied by F. Hence 4NF can be considered as a generalization of the BCNF that applies to relation schemes with multivalued dependencies.

1.3.3 Two Approaches for Database Design

Two approaches have been used for logical design of database schemes: analysis and synthesis. The former checks implication-related problems. The latter involves the generation of relation schemes in desired normal forms.

One of the objectives of analysis is to eliminate redundancy from a given set of data dependencies. Several algorithms have been developed for the solution of many implication-related problems, such as finding the closure and dependency basis, reducing redundancy of data dependencies, and testing the logical implication of new FDs and MVDs.

Synthesis is essentially concerned with the following problem: Given a universe of a database scheme and a set of data constraints, decompose the universe U into a set of relation schemes satisfying various normal forms with respect to these data constraints. Several algorithms for the generation and recognition of normalized relation schemes are briefly reviewed in Chapter 2.

Chapter II

REVIEWS

2.1 INTRODUCTION

The objective of this thesis is not to develop algorithms but to develop and automate a graphical model. No details are given about the algorithmic aspects. Instead, in Section 2.2, we just briefly mention some of the major algorithms for analyzing data dependencies and generating relations in normal forms. Section 2.3 reviews in greater details three approaches for expressing and testing the design problems. They are the graph-based approach, the logic-based approach and the data-structure-based approach. Lastly, in Section 2.4, we describe the logic-based database design tool RED1, the only system we know of for solving similar problems.

2.2 ALGORITHMIC FOUNDATION FOR LOGICAL DATABASE DESIGN

Many algorithms have been developed for the analysis of data dependencies and for the recognition and generation of normalized relation schemes.

Problems for analysis are often transformed into implication-related problems. The latter are solved by

algorithms for computing the dependency basis of the left-hand set of attributes of the dependency to be tested. These algorithms differ in their computational complexities. The best algorithm for the MVD implication problem is almost linear in time [GALI82]. Other linear-time algorithms for computing the dependency basis are given in [BEER80] and [MAIE81].

There exist several synthesis algorithms in the literature, one for the generation of relation schemes in 3NF [BERN76] and two decomposition algorithms [MAIE83, ULLM82] for relations in 4NF and BCNF.

The computational complexities of some algorithms for testing implication problems for a set of FDs, MVDs and JDs are reported in [MAIE81]. The major result is that testing whether a JD is logically implied by a set of FDs and JDs is an NP-Hard problem [FISH83, MAIE81].

2.3 THREE APPROACHES FOR DEVELOPING DATABASE DESIGN TOOLS

In order to investigate the logical design of relational databases, tools are required for presenting the problems and implementing the algorithms. These tools can be classified according to three widely recognized approaches: the graph-based approach, the logic-based approach and the data-structure-based approach. They are briefly reviewed in the following subsections.

2.3.1 The Graph-based Approach

Several models based on graphs are introduced in the literature. They include the graph-theoretic model, linear graphs, derivation directed acyclic graphs, hypergraphs, join graphs and S-diagrams. Our descriptions include their structures, applications, developments, advantages and disadvantages.

2.3.1.1 The Graph-theoretic Model

The graph-theoretic model [AUSI83] is mainly used for the representation of functional dependencies by FD-graphs. An FD-graph G = <V,E> is composed of the following elements: (i) Simple nodes representing individual attributes. (ii) Compound nodes each representing a set of attributes on the left-hand side of the functional dependencies. (iii) Full arcs joining simple nodes or compound nodes to the nodes representing the attributes at the right-hand side of the functional dependencies. (iv) Dotted arcs joining compound nodes to their simple constituent nodes to represent trivial FDs.

Analysis and manipulation of these FDs are conducted by graph algorithms for: i) solving the membership and redundancy elimination problems and ii) finding the closure and minimal covering of an FD-graph.

There are some advantages of this model. First, graph manipulation can be done by existing known graph algorithms. Secondly, the implementation does not require complex data structures. The weaknesses are that the construction of the FD-graph is not a straightforward transformation (or translation) of the set of FDs and its application is limited to functional dependencies only.

2.3.1.2 The Linear Graph Model

In this model, functional dependencies are represented by eight binary relations: dependency, equipotence, dissidence, completion and their respective dual relations. All eight binary representations can be expressed in terms of incidence matrices [NAMB83].

The dependency relation can directly exhibit the functional dependencies that are implied by the given functional dependencies. The starting point in its generation is the construction of an attribute relation. This relation is represented by a matrix of an order equal to the number of all possible subsets of the universe of attributes.

This model also has its drawbacks. First, with a large universe, the generation of binary relations using linear graph algorithms is not practical, especially for the elimination of redundant dependencies. Secondly, these

representative relations have to be stored permanently. Lastly, the application of this model is limited to functional dependencies only.

2.3.1.3 The Derivation Directed Acyclic Graph (DDAG)

A DDAG is a directed graph without loops. Each node is labeled with an attribute name. The DDAG is constructed recursively as described below:

1. Any set of unconnected nodes with labels from the universe is a DDAG.
2. Let H be a DDAG that includes nodes v1, v2, ..., vk with labels A1, A2, ..., Ak and let A1 A2 ... Ak --> C Z be an FD. Form H' by adding a node u labeled C and directed edges (v1,u), (v2,u), ..., (vk,u) to H. H' is a DDAG.
3. Nothing else is a DDAG.

The DDAG is mainly used for the representation of functional dependencies [MAIE83]. The membership problem is solved by constructing the F-based DDAG for X --> X+ and then testing whether or not those nodes for the attributes on the right-hand side exist in the DDAG.

One drawback of this model is that consideration of data dependencies other than FDs is not trivial. Its major defect is that interpretation of a DDAG is not unique, i.e., the same DDAG may represent two different sets of functional dependencies.

2.3.1.4 Hypergraphs and Join Graphs

A hypergraph is a graph-theoretic concept used to represent a database scheme. It is similar to an ordinary undirected graph, except that edges are arbitrary nonempty sets of nodes, rather than just doubletons.

Definition 2.1. A hypergraph H is a pair (N,E), where N is a set of nodes and E a set of hyperedges (or, briefly, edges) which are nonempty subsets of N, such that every node in N belongs to at least one edge in E.

There is a natural correspondence between relational database schemes and hypergraphs, with each attribute represented by a node and each relation scheme by an edge.

In the following definitions, a database scheme D over U is composed of relation schemes R1, R2, ... and Rp.

Definition 2.2. The complete intersection graph I for D is the complete undirected graph with nodes R1, R2, ... ,Rp. Each edge e = (Ri,Rj) has a label L(e) which is the intersection Ri $\cap$ Rj. An intersection graph for D is any subgraph of I formed by removing only edges from I.

Definition 2.3. Let G be an intersection graph for D and let A $\in$ U. A path e1, ... , ek from node Ri to node Rj in G is called an A-path if A $\in$ L(ei) for all k $\geq$ i $\geq$ 1.

Definition 2.4. An intersection graph G for D is a join graph if for every pair of nodes Ri, Rj in G containing an attribute A are connected by an A-path. A join tree is a join graph that is a tree.

It has been shown that it is a desirable property that the join graph of a database scheme is a tree [MAIE84]. Database schemes with such a property are called a-acyclic.

The major advantage of this model is that it can be used not only for manipulating dependencies, but also for query optimization, acyclicity recognition and determination of database coverings [CHEU86, ZHU84]. One disadvantage is that the hypergraphs become complicated for viewing for large databases.

2.3.1.5 S-diagrams

It has been shown [AROR81] that FD, MVD and several other types of data dependencies form a class of dependencies called root-dependency. An S-diagram is basically a hypergraph which describes root-dependencies. Each S-diagram represents an instance of a relation. Four conditions can be tested on an S-diagram, namely the L-T condition (loop-tuple) the C-condition (completeness), total C-condition and partial C-condition [AROR81]. Various root-dependencies can be determined by checking whether the diagram satisfies one or a combination of these conditions.

The main advantage of this graphical model is that it can represent a larger group of data dependencies. The main disadvantage is that the complexity of the graphical representation depends on the usually very large number of relation instances.

2.3.2 The Logic-based Approach

The syntactic similarity between formulas in propositional logic and data dependencies in relational databases was first pointed out by Fagin [FAGI77]. Since then it has been investigated extensively in the literature.

2.3.2.1 Propositional Logic and Data Dependencies

It can be shown that FDs and MVDs are equivalent to formulas in propositional logic [SAGI79]. This equivalence provides new techniques for proving properties of FDs and MVDs, and for solving the membership or implication-related problems. Efficient algorithms were found by Sagiv [SAGI80] and Galil [GALI82].

Equivalence Theorem: Let sig be a data dependency and SIG be a set of data dependencies. There exists an equivalence between the following two statements:

(i) sig is implied by SIG.

(ii) sig is a logical consequence of SIG.

The syntactical equivalence between the formula 'A1 and A2 ... and Aq ==> B1 and B2 and B3 ... and Bp' and a functional dependency is quite apparent. We can apply the inference rules for functional dependencies in order to obtain inference rules for formulas of the form X ==> Y.

For example, the transivity rule for FD can be expressed by the following logical formula:

If X ==> Y and Y ==> Z, then X ==> Z.

Such results for functional dependencies can be extended to multivalued dependencies.

Let U be the set of propositional variables. Let X, Y and Z be sets of propositional variables whose union is U and where Z is the complement of X Y in U. The formula X ==> Y + Z is true if either X is false or Y is true or Z is true. X ==> Y + Z corresponds to X ->-> Y.

The above example indicates that any multivalued dependency is transformable to a formula in propositional logic.

The equivalence between propositional logic and data dependencies implies that logic can be used for solving logical database design problems. It can also be used for query optimization and evaluation, ensuring data integrity, implementing logical query languages and several other problems.

A good review of the relationship between logic and databases is given in [GALL84b]. More details about research in this area can be found in [GALL78, GALL81a, GALL83, GALL84a, JACO82, KOWA79, KUNI82, ULLM85].

2.3.2.2 PROLOG for Database Design

The programming language PROLOG provides a convenient tool for database design. It is a high level, non-procedural language with a simple syntax. A PROLOG program consists of two parts: i) Facts that constitute the knowledge-base. ii) Rules correspond to logical implications that manipulate the facts in order to derive other facts.

The facts are quite similar to a particular representation of relations in a database. Relations correspond to functions. A tuple in an instance of a database corresponds to a set of atomic values for a function in a Horn clause [KOWA79].

The following example shows the rules of a PROLOG program to find the closure and implication of FDs. In this program, ":-" and "," stand for "if" and "and", respectively.

Example 2.1:

```
FD(X,Y) :- Fdep(X,Y).
FD(X,Y) :- Closure(X,CL) , InsideList(Y,CL).

Closure(X,CL) :- Fdep(X,Y),
                 Concatenate(X,Y,Z),
                 RemoveDuplicate(Z,U),
```

```
                    Closure(U,CL).

Closure(X,CL) :- Fdep(W,Y),
                 InsideList(W,X),
                 NOT InsideList(Y,X),
                 Concatenate(X,Y,Z),
                 RemoveDuplicate(Z,U),
                 Closure(U,CL).
```

The above rules use facts of the form Fdep(<lhs>,<rhs>) where <lhs> and <rhs> are the attributes on the left and right hand sides of the FDs, respectively. A query about the implication of an FD given the set of FDs is of the form: ?- Fd(<lhs>,<rhs>). Finding the closure of a set of attributes is solved by the query:

```
?- Closure(<attributes>,X).
```

2.3.3 The Data-structure-based Approach

In this approach data dependencies are represented by a particularly designed and efficient data structure. Several algorithms are developed for the manipulation of the representative data structures. Their efficiencies and computational complexities depend on the associated data structures.

A typical data-structure for the membership algorithm for functional and multivalued dependencies is described in [HAGI78].

2.4 RED1: A LOGIC-BASED SYSTEM FOR DATABASE DESIGN

Though there have been a lot of algorithmic results about the design problems mentioned above, there seems to exist only two systems [BJOR83, CERI86] which automate the processes for analysis and synthesis. In this section, we present RED1 developed by Bjornstedt.

RED1 has the following objectives:

a) To assist in the development of a prototype of a relational database.

b) To execute queries and transactions for testing purposes.

c) To assist in the data dependency analysis and normal form testing.

RED1 is basically a PROLOG program that consists of:

(i) The PROLOG functions and utilities. These are called the "rules".

(ii) The PROLOG database or knowledge-base. This is a set of facts to be manipulated by the rules.

The facts in RED1 are:

a) Relations and their attributes.
e.g., likes(person, color), drink(person,liquid).

b) Data Dependencies.

e.g., fundep(person,address) or multdep(person,courses).

c) Integrity constraints.

d) A tuple representing the Universe of attributes.

Universe(person, color, liquid, address).

e) Tuples representing transactions.

2.4.1 Functional Organization of REDI

REDI is functionally divided into two parts: the database management system (DBMS) and the constructive part.

i) The Database Management System

This part allows the database designer to create and define data types, domains of data types and attributes, relations on attributes and finally integrity constraints on relations. In addition, a query language with procedural capabilities for query and transaction manipulations is available. Using these utilities, the designer has the means for prototyping a small relational database.

ii) The Constructive Part

This part consists of several utility procedures used for the study of some logical design aspects. Functional and multivalued dependencies are asserted by the operators "-->" and "->->". The designer can query REDI about several implication-related problems, such as finding the closure

and dependency basis of a set of attributes, and testing whether an FD or an MVD is logically implied by an existing set of data dependencies.

Problems related to decomposition and synthesis of normalized relation schemes can also be solved by RED1. A command called "synthesize" produces a set of relation schemes in 3NF. The designer can test whether a relation scheme is in 4NF.

Chapter III

AN EXTENDED PETRI-NET MODEL FOR THE REPRESENTATION OF DATA DEPENDENCIES AND RELATION SCHEMES

3.1 INTRODUCTION

In this chapter we describe a graphical model for the representation of relation schemes and the data dependencies imposed on them. This model was recently introduced by Cheung [CHEU85]. It is the backbone of our graphical system to be described in Chapters 4 and 5. Many membership-related and acyclicity problems can be solved by simple mathematical computations based on the model.

In Section 3.2, we introduce some basic definitions of ordinary Petri-nets. Sections 3.3 and 3.4 introduce the representations of functional, multivalued and relation schemes within the model. In Section 3.5, we state the advantages of our model and compare it with other graphical methods.

3.2 FUNDAMENTALS OF PETRI-NETS

Many variations of Petri-nets have been introduced in the literature. To describe our model, it is enough to consider the simplest one, namely the basic Petri-net or simply called Petri-net [PETE77].

Definition 3.1. A basic Petri-net (BPN) is a quadruple N(P,T,F,B), where P is a set of places ($P \neq O$) represented by circles, T is a set of transitions ($T \neq O$ and $P \cap T = \emptyset$), represented by single bars,

F: P x T -> N, a forward incidence function, where N = {0,1} and

B: P x T -> N, a backward incidence function, where N = {0,1}.

If we order the sets P and T, F and B can each be represented by an m x n matrix, where m is the number of places and n the number of transitions. There is an arc from place pk to transition ti iff F(pk,ti) = 1 and there is an arc from transition tj to place pl iff B(pl,tj) = 1.

Definition 3.2. Every place pi $\in$ P has a non-negative number of tokens. The vector M representing the distribution of tokens is called a marking of the Petri-net. That is, M = (M(pi)), i=1,2 ...,m, where M(pi) denotes the number of tokens at place pi.

Definition 3.3. For a given Petri-net and a transition t $\in$ T, PRE(t) = {p $\in$ P / F(p,t) $\neq$ O } is the set of input

places of transition t and POST(t) = {p ∈ P / B(p,t) ≠ 0} is the set of output places of transition t.

Definition 3.4. Let sig be a sequence of transitions t1 t2 ... tk. sig is called a legal firing sequence starting from M if and only if there exists a sequence of markings M ..... M* such that:

$$M \xrightarrow{t1} \cdots\cdots\cdots \xrightarrow{tk} M^*$$

The marking M* is said to be reachable from M by firing sig.

Definition 3.5. A transition t of a Petri-net is enabled under a given marking M if and only if for every p ∈ PRE(t), M(p) ≥ F(p,t). Firing of a transition t ∈ T is defined by the transformation of the enabling marking M into another marking M' such that:

$$p \in P,\ M'(p) = M(p) + (B(p,t) - F(p,t))\ sig.$$

Note that the number of marked places and the number of tokens after firing may be greater, smaller or the same.

Definition 3.6. The forward marking class is the set of markings reachable from the initial marking M by a legal firing sequence.

3.3 PETRI-NET REPRESENTATION OF FUNCTIONAL DEPENDENCIES

In the following, we describe how a set of FDs can be uniquely represented by the transitions of a Petri-net. We then develop its mathematical foundation.

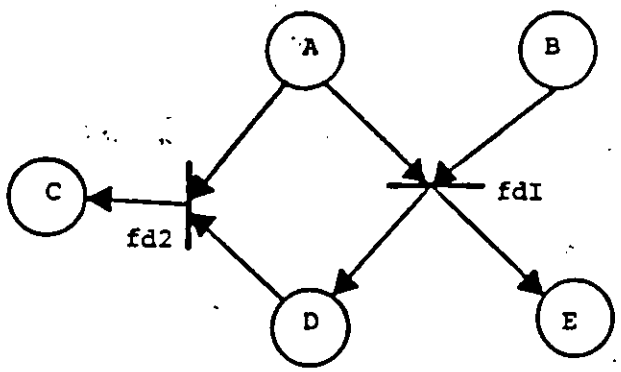
Rules for creating an Extended Petri-net for a set of FDs

Let U = {Aj}, for j=1 to q, be the universal set of attributes and F = {fj: Xj --> Yj, Yj $\subset$ U}, for j=1 to n, be a set of FDs defined over U. An Extended Petri-net N(P,T,F,B) can be created as follows:

a) For each attribute Aj $\in$ U, create a place called attribute-place, or simply A-place with label Aj.

b) For each FD fj: Xj --> Yj in F, create a transition with label tj, such that PRE(tj) = Xj and POST(tj) = Yj.

Example 3.1: The set of FDs {fd1: A B --> D E, fd2: A D --> C} can be represented by the following Extended Petri-net:



An A-place p is said to be reachable from an initial marking M if there exists a sequence of firings starting from M such that p obtains a token.

The EPN model for the representation of FDs is a modification of the UF-based Petri-net model originally proposed by Cheung (CHEU86). In the EPN model, we omitted the third rule in the construction of the Petri-net for the representation of FDs.

Matrix Equation for an Extended Petri-net

For a given Extended Petri-net, two consecutive markings after firing a transition are related by the following matrix equation:

$$A' = A + C.u$$

where

A = col(a1, a2, ..., aq), A' = col(a'1, a'2, ..., a'q), aj and a'j are the numbers of tokens in the A-place Aj.

C = B(p,t) - F(p,t)

B(p,t) = 1 if p $\in$ Yj and t = tj for some j

0 otherwise.

F(p,t) = 1 if for p $\in$ Xj and t = tj for some j

0 otherwise.

and

u = col(t1, t2, ... , tn), ti=1 if it is enabled otherwise ti=0.

Correspondence between FDs and Petri-nets

In the theory of functional dependencies, an FD is said to be 'applicable' to a relation scheme if it is valid for any relation instance of that scheme. One FD can be 'applied' to 'imply' another. These 'applicable', 'applied' and 'imply' concepts for FDs have corresponding concepts in an Extended Petri-net, as shown in Table 3-1.

Functional dependency	Petri-net
a) applicable	enabled or firable
b) applied	fired
c) F |= X --> Y	Y is reachable from X by executing a sequence of firings.

Table 3-1. Correspondence between FDs and Petri-nets.

In particular, correspondence c) states that the implication problem for a set of functional dependencies F can be solved as a submarking reachability problem for its Extended Petri-net.

Algorithms for computing the closure of a set of attributes and for removing redundancy from a set of FDs have been developed [CHEU86]. The details are omitted here because, as mentioned before, it is not the objective of this thesis to develop or to describe algorithms.

3.4 EXTENDED PETRI-NETS (EPN) FOR THE REPRESENTATIONS OF MUTLIVALUED DEPENDENCIES AND RELATION SCHEMES

This section extends the above Petri-net model to include the representations of multivalued dependencies and relation schemes.

In this extension, each MVD is represented by an MVD-transition, denoted by a double bar. The left-hand side and right-hand side of the MVD are represented by the set of input places and the set of output places, respectively. Figure 3-1 shows the EPN representation of the MVD: A B ->-> C D.

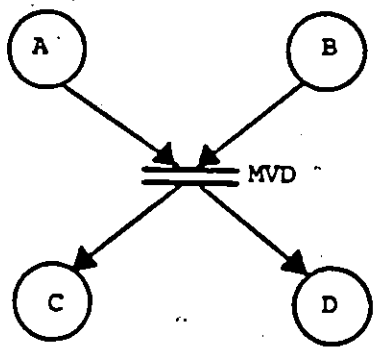


Figure 3-1. EPN representation of A B ->-> C D.

Each relation scheme is represented by an R-transition denoted by a square. The edges emanating from an R-transition are undirected and connected to those attributes of this scheme. The totality of the R-transitions

may be used to represent a join dependency defined on the database scheme. Figure 3-2 shows the graphic EPN representation of the JD: jd1[A B, B C D, A D E].

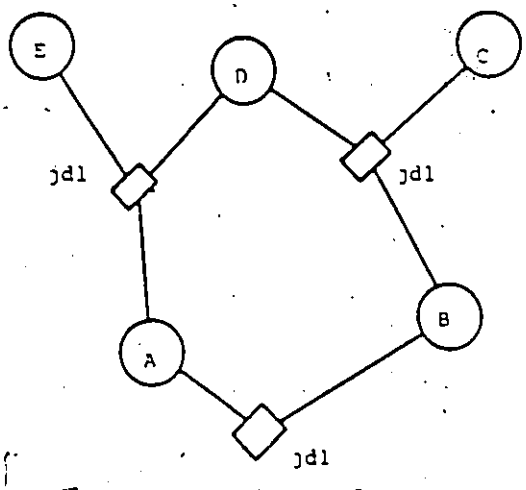


Figure 3-2. EPN representation of jd1.

Example 3.1.

Figure 3-3 shows the EPN representation of the following set F of FDs, MVDs and JDs:

```
F = { fd1:    A    -->  B C D E
      mvd1:   B C ->-> A I
      jd1:    {A B C D, C D E, B D I}  }.
```

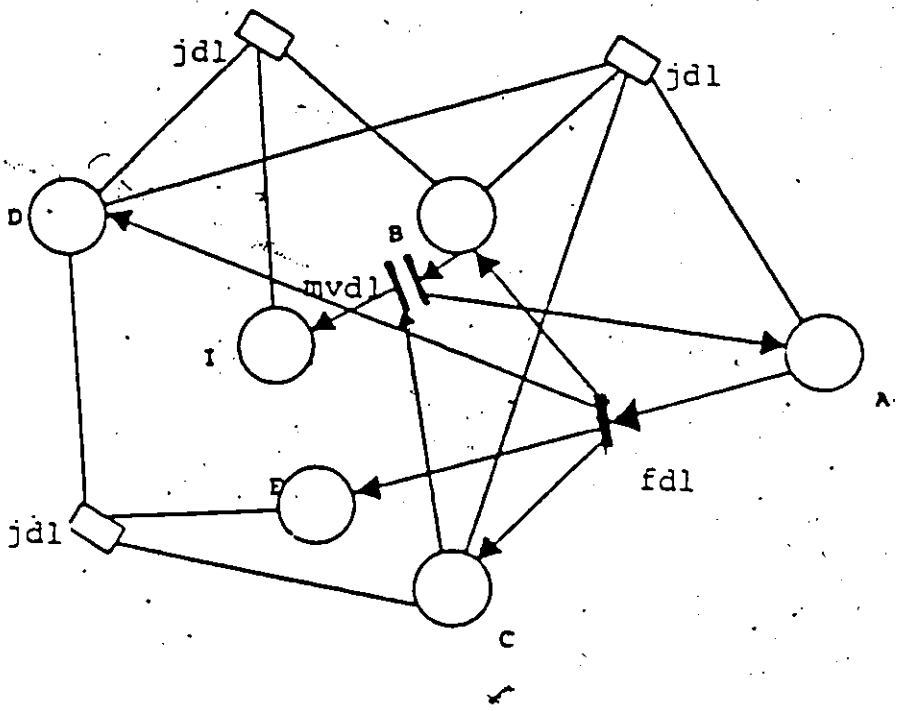


Figure 3-3. EPN representation of Example 3.1.

3.5 COMPARISON WITH OTHER GRAPHICAL MODELS

In this section we present two examples. The first compares the EPN model with the derivation directed acyclic graph (DDAG) and the graph-theoretic models. The second compares the EPN model with a hypergraph.

The following criteria are used in our comparison: (i) How simple is it to construct the graphic representations manually? (ii) How clear and appealing to viewers' intuition are the representations? (iii) How efficient are the representations for computer implementation?.

Example 3.2.

Consider the following set of FDs:

```
F = { A     --> F B C,
      C     --> D,
      F B D --> H,
      B D   --> E }
```

Figures 3-4, 3-5 and 3-6 show the FD-graph, DDAG and EPN representations of the given FDs, respectively.

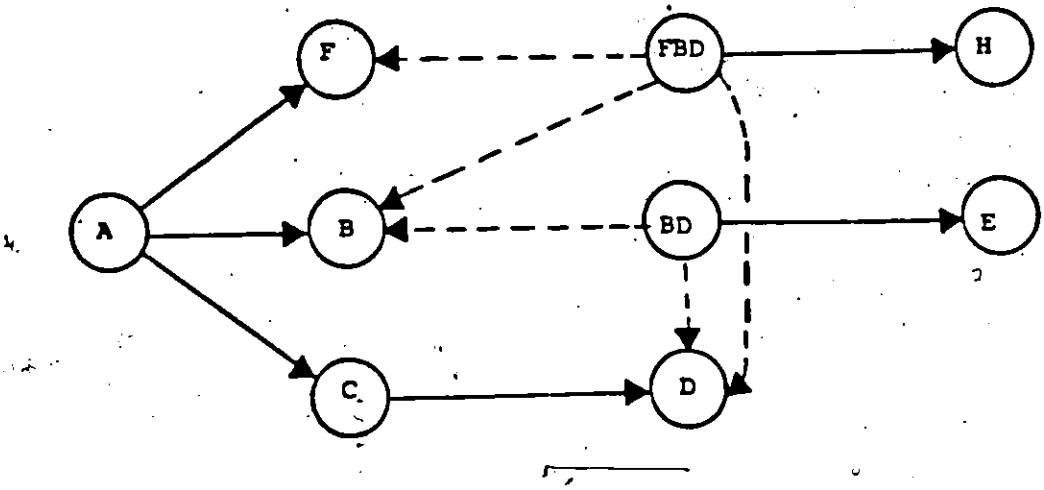


Figure 3-4. FD-graph representation of F in Example 3.2.

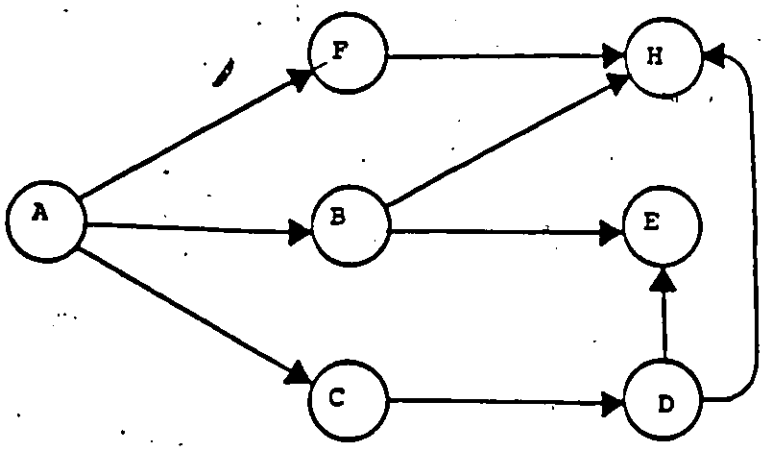


Figure 3-5. DDAG representation of F in Example 3.2.

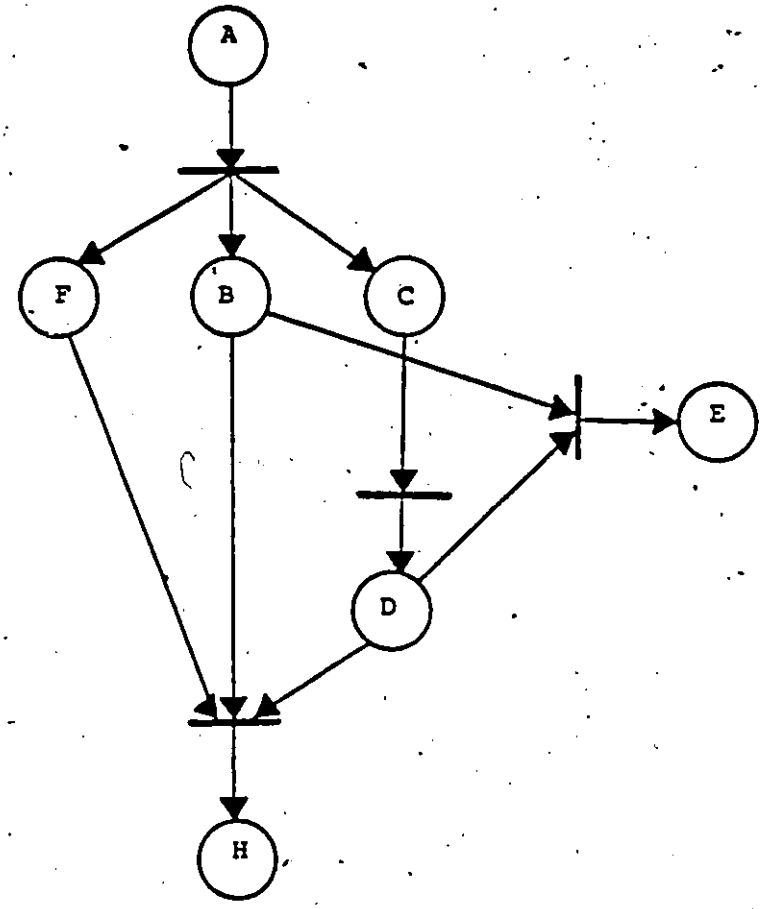


Figure 3-6. EPN representation of F in Example 3.2.

Simplicity of manual construction procedures:

The procedures for the construction of the FD-graph, DDAG and the EPN models are given in Chapters 2 and 3.

It is clear that an EPN representation can be easily obtained from the given set of FDs; whereas the construction of the FD-graph requires more complex steps and that of DDAG requires an exhaustive recursive operation.

Clarity and appeal to viewers' intuition:

An EPN representation is clear and unique. By uniqueness, we mean that given a set of FDs, we can get only one EPN representation, and from it, we can get the given set of FDs (Figure 3-5). By intuition, the viewer can understand clearly the relationships of the attributes within the FDs. In the case of DDAG, the representation of the given set of FDs is not unique. (For instance, one cannot know if the FD: B --> H E is in the given set F or not). In the case of the FD-graph representation, the user becomes confused by with the dotted arcs, but the original set of FDs can be uniquely derived.

Efficiency of computer implementation:

Although the three models are all quite simple for computer implementation, their complexities are quite different. The FD-graph requires more arcs (full or dotted) and nodes. In the case of the DDAG and EPN, the number of nodes is limited to the number of elements in the universe of attributes. Therefore, storage space for the graphical data in data structures is easier to control.

Example 3.3.

Consider the database scheme: D = (A B C, A E C, A E F, E D C).

The hypergraph and EPN representations are shown in Figures 3-7 and 3-8, respectively.

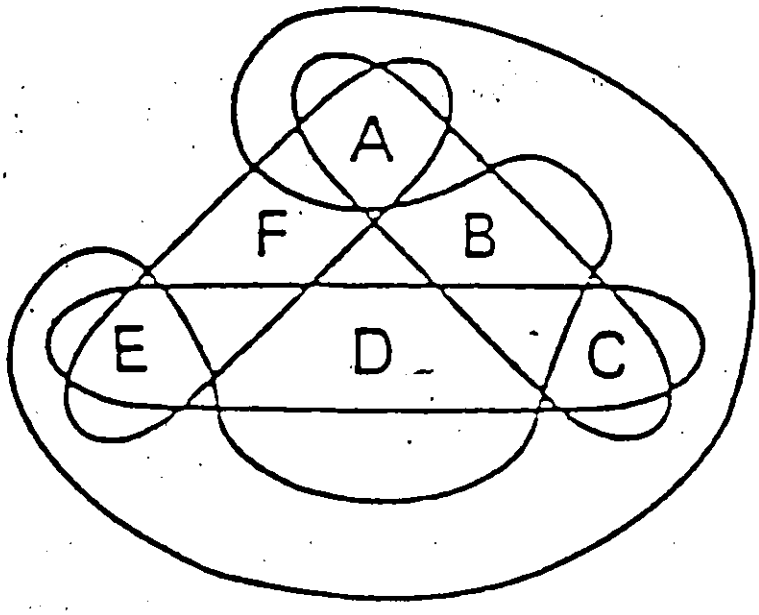


Figure 3-7. Hypergraph representation of D in Example 3.3.

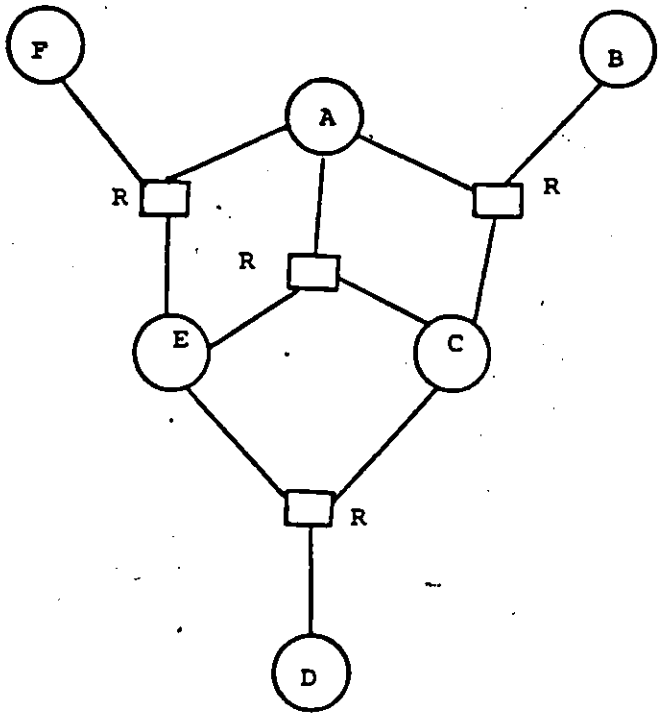


Figure 3-8. EPN representation of R in Example 3.3.

The procedures for manually constructing both representations are simple. Hypergraphs are difficult and confusing for viewing and drawing. It is also inefficient for automation because of their irregular shapes. The EPN representation is clear and appeals to the viewers' intuition. Its computerization is easy since it contains only regular shapes such as circles, squares and straight lines.

Table 3-2 summarizes the features of the EPN, hypergraph, DDAG and the graph-theoretic representations.

	Manual construction	Appeal to intuition and clarity	Computerization
EPN	• simple	• appeals to viewers' intuition. • representation is unique.	• easy to implement.
Hyper-graph	• simple	• difficult to view and draw. • representation is unique.	• inconvenient because of its irregular shapes.
DDAG	• complex (recursive)	• confusing to the user. • representation may not be unique.	• easy to implement.
Graph-theo-retic	• complex	• complex and unclear representations for large set of FDs. • representation is unique.	• inconvenient for implementation

Table 3-2. Comparison among EPN, hypergraph, DDAG and grapgh-theoretic representations.

Chapter IV

DETAILED DESCRIPTION OF GLORD

(from a user's point of view)

4.1 INTRODUCTION

In this chapter we describe in details the functions of our Graphic system for LOgical Relational database Design (GLORD) from a user's point of view. Section 4.2 explains how to start GLORD under the MacIntosh microcomputer. Sections 4.3, 4.4 and 4.5 describe the functions and background information of several subsystems of GLORD: the graphic support subsystem, the subsystem for designing and testing data dependencies and normal forms and the HELP subsystem. Section 4.5 compares the functions of GLORD and RED1, an existing logic-based system for similar purposes. Descriptions of the menus and their entries in the GLORD subsystems are given in the user's guide (Appendix A).

4.2 STARTING GLORD UNDER THE MACINTOSH MICROCOMPUTER

GLORD is an application program of 36 Kbytes to be executed on an Apple Macintosh microcomputer. A memory of 512 Kbytes and a dedicated printer are needed. To apply GLORD, the user should be familiar with the operations in the Macintosh environment. The following paragraph describes how to start GLORD.

Once the diskette containing GLORD and the Macintosh Pascal interpreter is loaded, the main Macintosh system menu is displayed and the user should first open GLORD by clicking twice at its ICON. Once GLORD is opened, the user should select entry GO from Macintosh Pascal menu RUN. The welcome message of GLORD is then displayed.

In the following subsections, the menus and entries of the three subsystems are listed. Further details about the operations of the entries are given in Appendix A.

4.3 SUBSYSTEM FOR GRAPHIC SUPPORT

The functions of the graphic support subsystem of GLORD allow the designer to graphically create or alter existing database schemes.

This subsystem has the following menus and entries:

- menu DATABASE MODIFICATION:

  - ADD ARC

- ADD FD
- ADD MVD
- ADD SCHEME
- ADD ATTRIBUTE
- DELETE ARC

• menu UTILITIES:

- DISPLAY DATABASE (display the Extended Petri-net representation of the database scheme)

• menu FILE UTILITIES:

- SAVE PICTURE (save the graphic representation in a file)

• others (refer to Appendix A for detailed description).

4.4 DESIGN AND TEST SUBSYSTEM

The core of GLORD consists of two menus each having a set of functions for assisting the logical design of relational databases:

a) menu ANALYSIS:

- IMPLICATION (test implication of FD or MVD)
- CLOSURE (compute closure and dependency basis)
- KEY (find or test a key for a relation scheme)

b) menu NORMAL FORMS:

- SYNTHESIZE 3NF (generate by synthesis relation schemes in 3NF)

- TEST 3NF (test whether or not a scheme is in 3NF)
- GENERATE BCNF (generate relation schemes in BCNF)
- TEST 4NF (test whether or not a scheme is in 4NF)
- GENERATE 4NF (generate relation schemes in 4NF)
- TEST ACYCLICITY (test if a database scheme is alpha-acyclic)

4.5 HELP SUBSYSTEM

GLORD provides a help function for each of the menus. A help message briefly explains the function of an entry in a menu and, if applicable, the format of the expected user's query. These help messages are stored in a file called 'MSGS' used by GLORD. Appendix B contains the list of all such messages.

4.6 COMPARISON OF GLORD AND RED1

In the following, we compare GLORD with RED1 (an existing tool for solving similar database design problems as described in Chapter 2) according to two aspects: user interface and scope (domain) of the system's functions.

User Interface:

GLORD offers a user-friendly interface that allows the user to create, alter, analyse and test database schemes in an interactive and graphical environment. In addition, it provides utilities for recording the graphics and for

offering help messages. In RED1, the interface with the user is not graphical. RED1 offers an interactive query/answer interface in which alteration of the database schemes is done separately from the design session. This may cause inconvenience to the user.

Scope of Functions:

GLORD solves the basic problems in the logical design of relational databases, namely the analysis of data dependencies and generation of normalized relation schemes. It accepts FDs, MVDs and JDs.

RED1 automates the logical design processes for relational databases. It also performs some database management system (DBMS) functions such as query optimization and transaction manipulations when testing the databases. However, it accepts only FDs and MVDs.

Table 4-1 summarizes the differences of the two systems.

	G L O R D	R E D 1
User Interface	• User-friendly and graphical.	• Query/answer, non-graphical.
Scope of functions	• Accepts FDs, MVDs and JDs. • Tests and generates all normal forms.	• Accepts FDs and MVDs only. • Performs DBMS functions. • Generates normal forms.

Table 4-1. Comparison between GLORD and RED1.

Chapter V

DETAILED DESCRIPTION OF GLORD

(from a system's point of view)

5.1 INTRODUCTION

In this chapter we describe the features of GLORD (a Graphic system for LOgical Relational database Design) from the system's point of view. In Section 5.2 GLORD is described as a subsystem of a larger system used for the design and analysis of problems which are formulated in terms of Petri-net-based models. In Section 5.3 we describe the hardware and software requirements and the files used in GLORD. Some programming aspects of GLORD are described in Section 5.4.

5.2 GLORD: AS A SUBSYSTEM OF A LARGER SYSTEM

GLORD is part of a user-friendly graphic system used for the design and analysis of application problems formulated by means of Petri-nets such as protocol verification and synthesis, database management, etc. GLORD consists of three subsystems: graphic support, database test and design and HELP. The graphic support subsystem may be used in all applications.

Figure 5-1 shows a global view of the interaction of the graphical subsystem with the other subsystems.

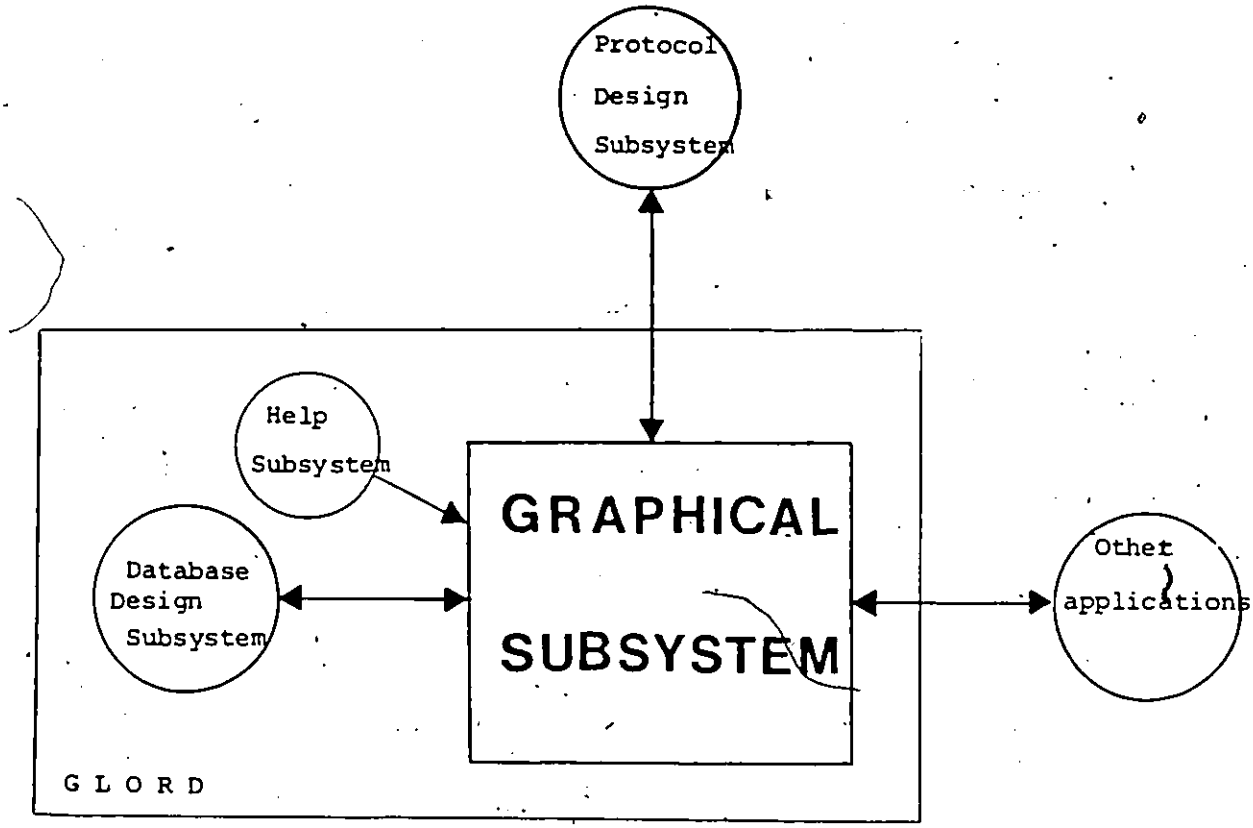


Figure 5-1. A global view of the subsystems.

After creating a new database scheme, a data structure containing its control information is stored. Some fields of the data structure are not used in GLORD, but they may be used in other subsystems. The reason for having a common data structure is to share the same graphic interface among all Petri-net-based subsystems. Figure 5-2 shows the data structure.

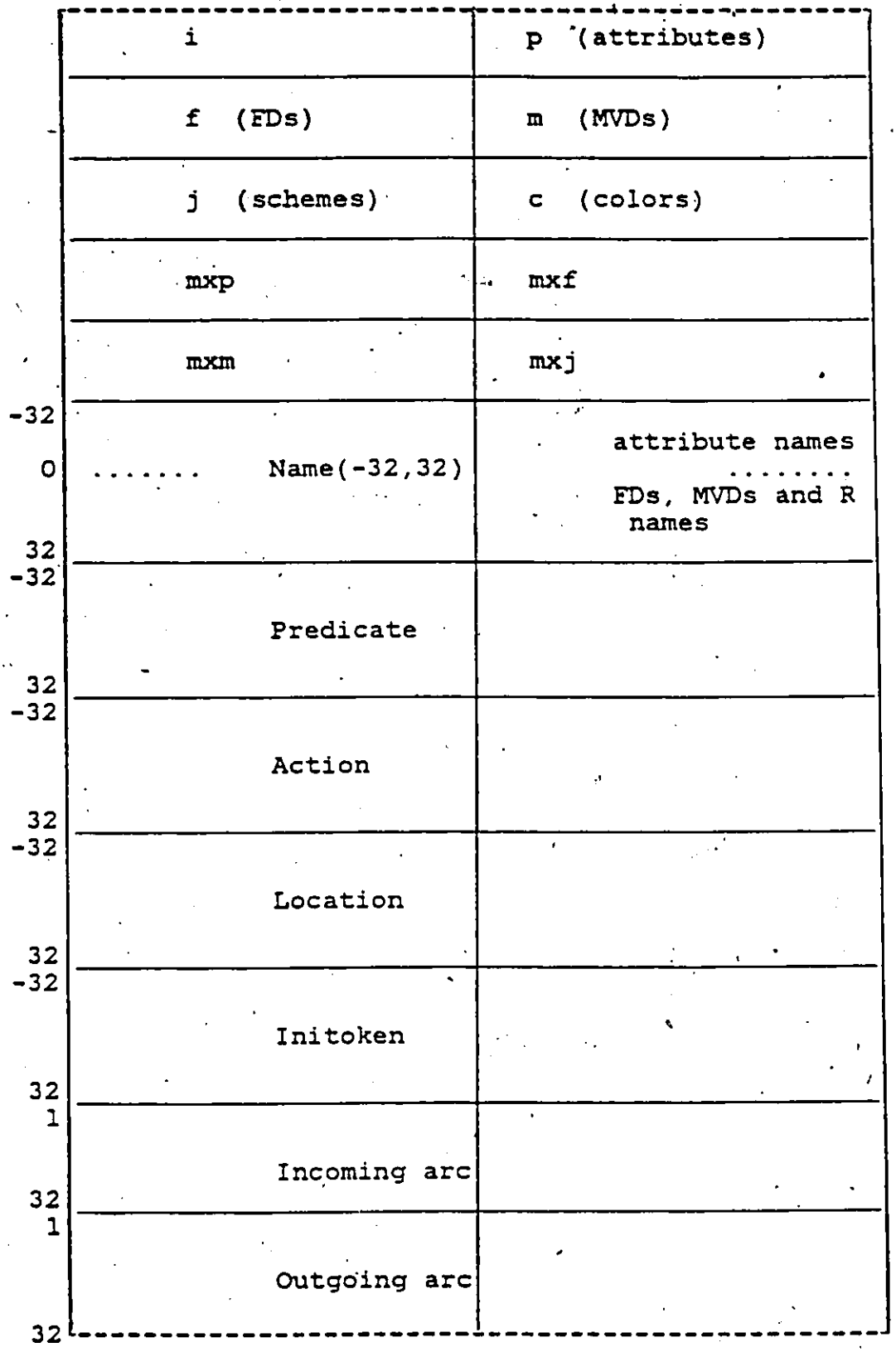


Figure 5-2(a). Data structure for database control information.

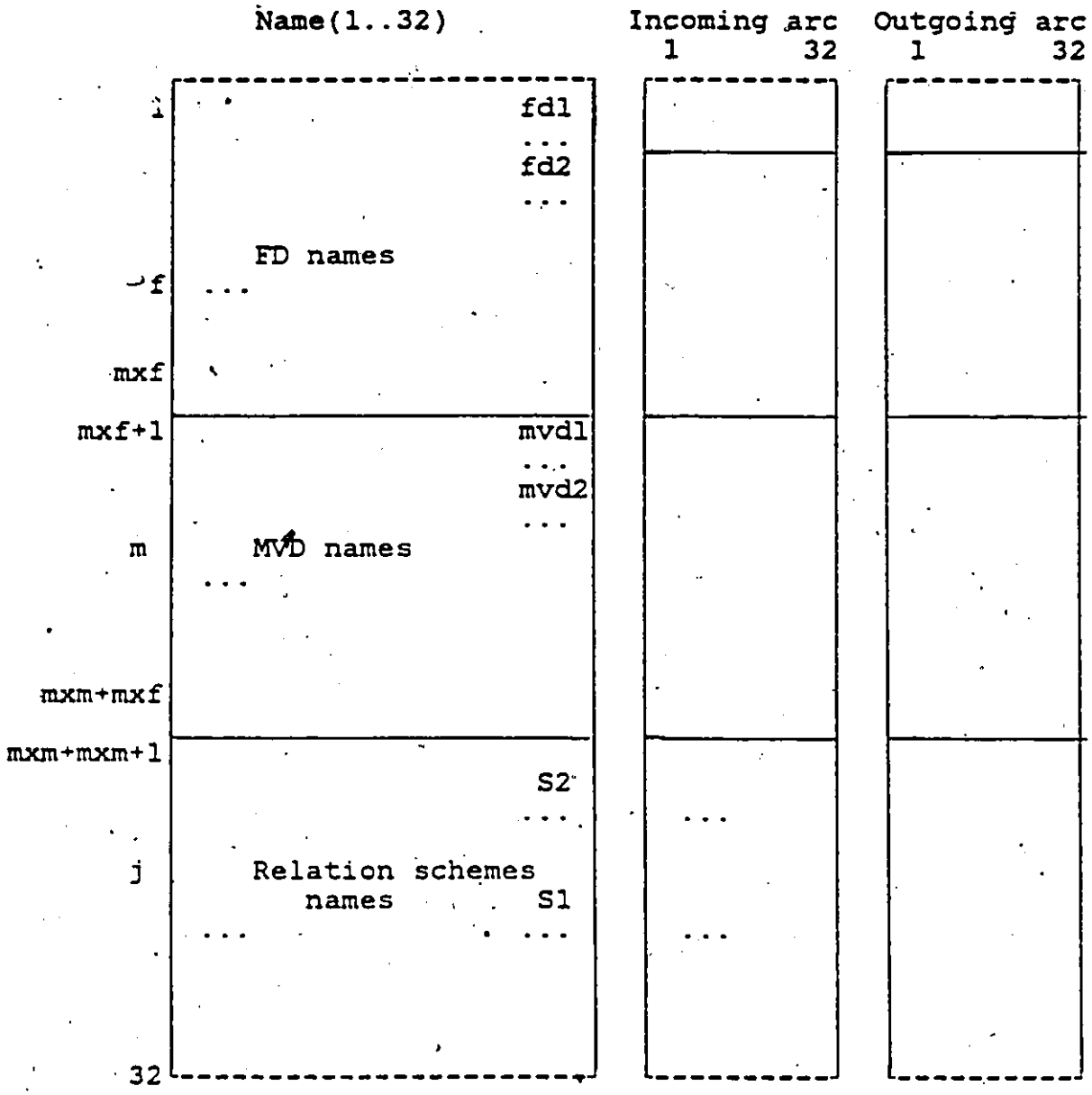


Figure 5-2(b). Data structure for database control information.

Table 5-1 describes the different fields of the data structure. Part (a) of the table shows their types and sizes. Part (b) describes their meanings.

```
ext_petri_net = record
  id                    : integer;
  p, f, m, j, c         : integer;
  mxp, mxf, mxm, mxj    : integer;
  name                  : array[-32..32] of string[16];
  predicate, action     : array[-32..32] of string[32];
  location              : array[-32..32] of point;
  inittoken             : array[-32..32 , 1..4] of integer;
  incoming, outgoing    : array[1..32] of longint;
end;
```

Table 5-1(a). Types and sizes of the fields of the data structure.

F I E L D	D E S C R I P T I O N
id	• identifier (initialized to 1) for use by other subsystems.
p, f, m, j	• number of attributes, FDs, MVDs and relation schemes, respectively.
c	• number of colors in case of colored Petri-net. (Not used in GLORD).
mxp,mxf,mxm,mxj	• maximum allowed number of attributes, FDs, MVDs and JDs in the database, respectively.
name	• array for storing names of attributes, FDs, MVDs and relation schemes.
predicate, action	• arrays describing the predicates and actions on every type of transitions. (Not used by GLORD).
location	• array for storing the location of every object of the database on the screen.
inittoken	• array for storing the distribution of tokens among the places. (Not used by GLORD).
incoming,outgoing	• arrays for storing the attributes on the left-hand and right-hand sides of FDs and MVDs, respectively.

Table 5-1(b). Meanings of the fields of the data structure.

5.3 HARDWARE/SOFTWARE REQUIREMENTS AND FILE ENVIRONMENT

GLORD is a Pascal application program of 36 Kbytes to be executed on an Apple MacIntosh microcomputer. A memory of 512 Kbytes and a dedicated printer are needed.

A utility program, called 'loadmessages', is developed along with GLORD and used to generate a message file called 'MSGS'. It contains all help, query and answer messages generated by GLORD. It should be opened for use once GLORD is initiated.

System software required by GLORD includes the MacIntosh Pascal interpreter, the built-in graphic procedures and Macpaint.

The files used in GLORD are shown in Figure 5-3 and described in Table 5-2.

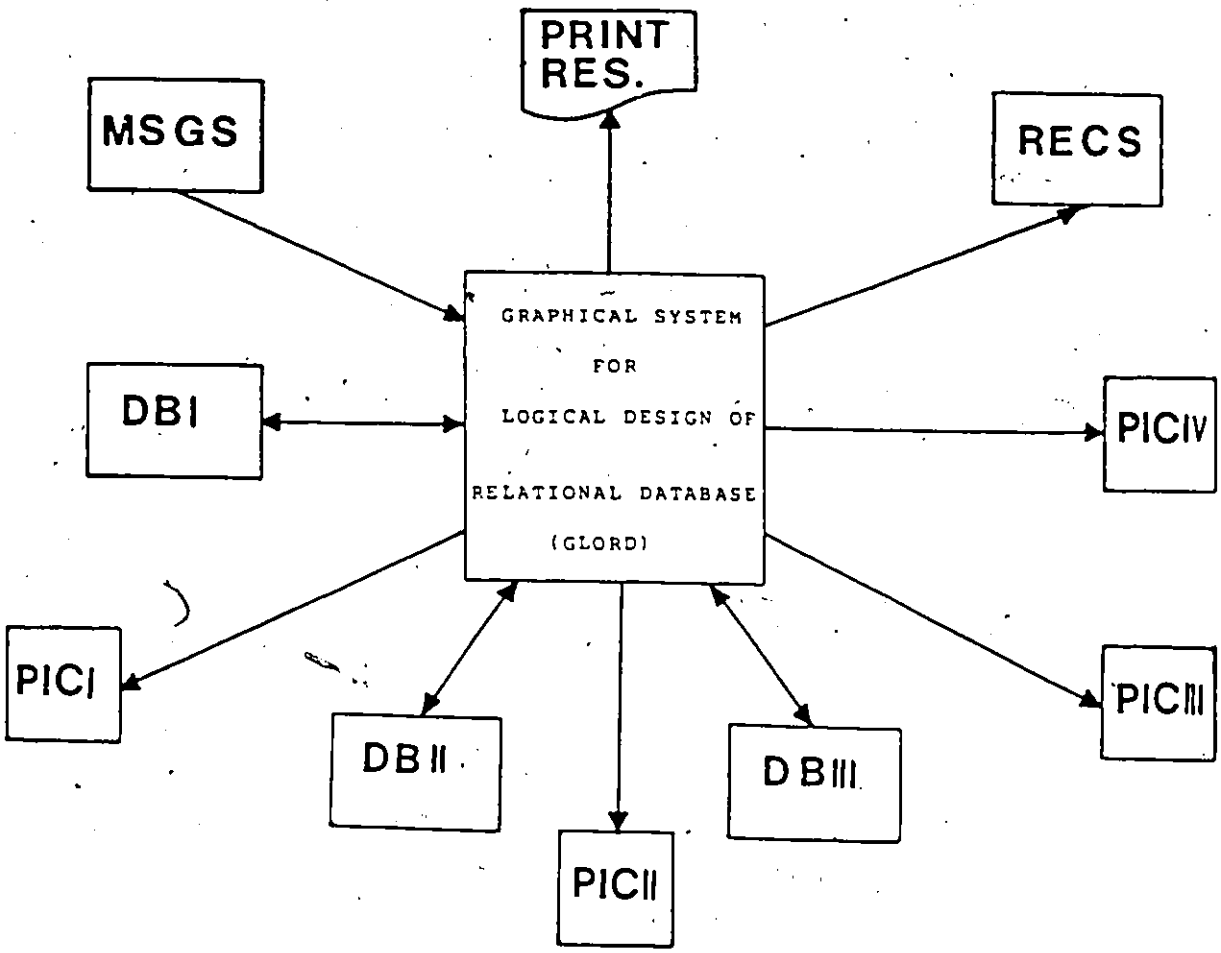


Figure 5-3. Files used in GLORD.

F I L E S	D E S C R I P T I O N
.DBI, DBII, ...	• Database files, each containing a structure describing a database scheme. Each about 8 Kbytes in size.
MSGS	• A message file associated with GLORD. Generated when 'loadmessages' is run. Each about 32 Kbytes in size.
PRINT RES.	• A hard-copy of user/system interactions.
PICI, PICII, .	• Stored database graphics. MacPaint can be used to open, display and print these files. Each about 7 Kbytes in size.

Table 5-2. Description of files used in GLORD.

5.4 DESCRIPTION OF GLORD

Figure 5-4 shows a skeleton pseudo-code of GLORD. The main program of GLORD contains a loop which takes different actions in response to the entries selected by the user from the menus. The procedure "Handle Menu" will call the appropriate functions or procedures to execute users' request.

The procedures and functions of GLORD are classified into two groups: one for graphic support and another for database design. Appendix C contains a more detailed description of these functions and procedures.

Implementation of GLORD requires extensive use of the Long Integer data type which is represented by 32 bits. For example, a relation scheme or a set of attributes is represented by a long integer: a '1' value at position 'i' means that the attribute numbered 'i' exists in the relation scheme. GLORD contains many utility procedures and functions for the manipulation of lists, such as inclusion, intersection, union, complementation, bit test and bit set. With the Long Integer data types, their manipulation becomes very easy and flexible.

```
GRAPHICAL
SUBSYSTEM     Graphic support functions

              List manipulation functions
DATABASE
DESIGN
SUBSYSTEM

              Database design and test functions
```

```
Procedure Handle Menu:
 case selection of:
  1:
  2:
  ....
 end Handle Menu;
 Main Program:
   LOOP
     Handle Menu;
   UNTIL exit from GLORD;
```

Figure 5-4. Skeleton pseudo-code of the GLORD program.

Chapter VI

EXAMPLES AND SOME CONCLUDING REMARKS

In this chapter we first present three examples showing the application of GLORD to the logical design of relational databases. Then, we make some concluding remarks.

6.1 EXAMPLES

Example 6.1

Consider the Suppliers and Parts database scheme from Date's well-known book [DATE85]. The universe of attributes is:

U = { pnum, pname, snum, sname, qty, status, city, weight, color}.

The set of data dependencies is:

F = { snum --> sname status city,
sname --> snum status city,
pnum --> city pname weight color,
snum pnum --> qty }

After using the appropriate entries of the menu DATABASE MODIFICATION, a file containing the structure describing the EPN representation is saved. Figure 6-1 shows the EPN representation of Example 6.1. It is displayed by using the entry DISPLAY DATABASE of the menu UTILITIES.

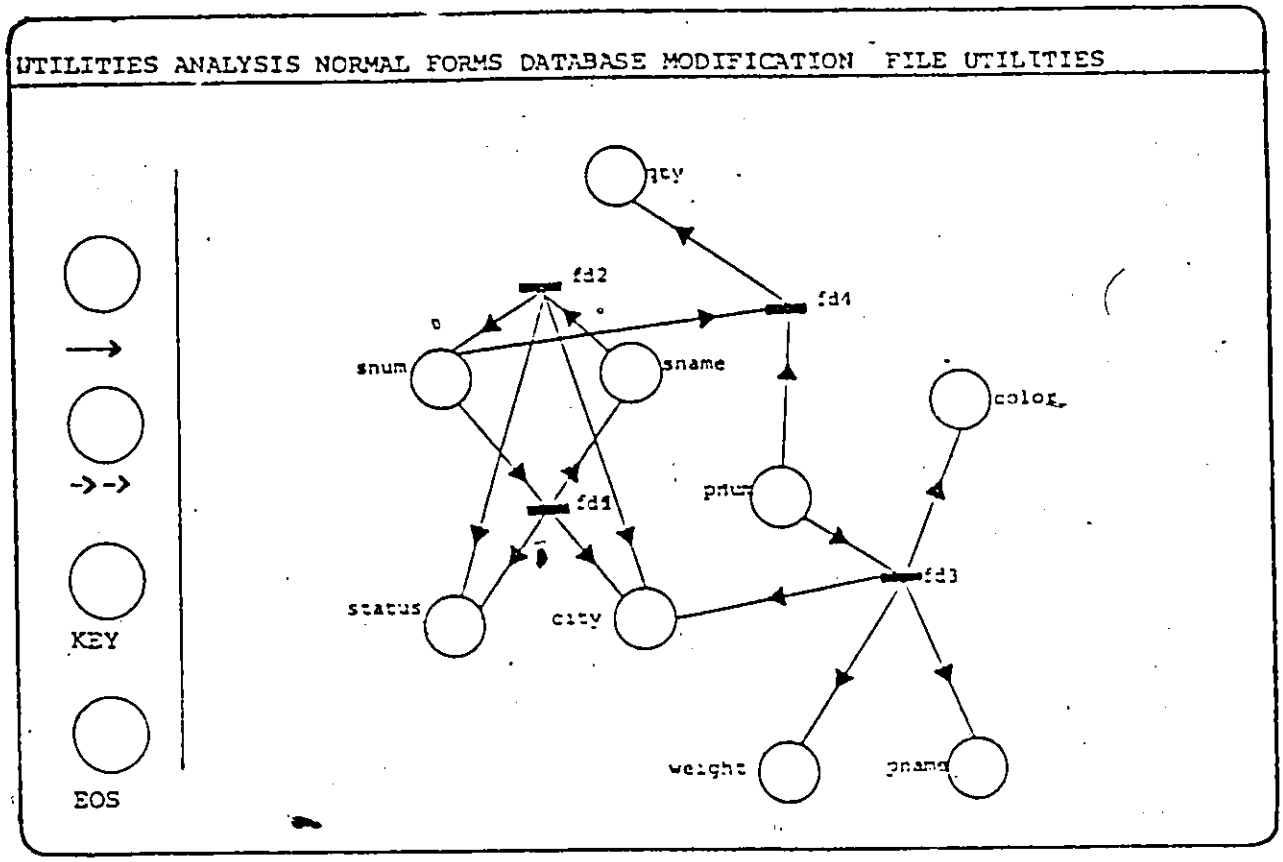


Figure 6-1. EPN representation of database in Example 6.1.

Following is the printed record of an analysis session. The name of the database, the universe and the data dependencies are printed by using entry PRINT DATABASE of the menu UTILITIES. Then all analysis queries (entered graphically) and their answers are printed. A graphical result of one query is also shown.

```
-----------------------------------------------------------------
--------------- GLORD DESIGN SESSION ----------------------------
-----------------------------------------------------------------

GIVEN : The database we design and test is: DBASE:EX1
-----------------------------------------------------------------
 The universe of attributes of this database is:
 snum sname status city pnum pname weight color qty
-----------------------------------------------------------------
 The data dependencies are:

 fd1 : snum --> sname status city
 fd2 : sname --> snum status city
 fd3 : pnum --> city pname weight color
 fd4 : snum pnum --> qty

-----------------------------------------------------------------
*************** ANALYSIS OF DATA DEPENDENCIES ********************
-----------------------------------------------------------------

Testing whether the following data dependency is logically implied:
snum --> pnum
Answer >
Not Logically implied.
-----------------------------------------------------------------
Testing whether the following data dependency is logically implied:
snum --> sname
Answer >
Logically implied.
-----------------------------------------------------------------
Testing whether the following data dependency is logically implied:
snum ->-> pnum,pname,color,qty,weight
Answer >
Logically implied.
-----------------------------------------------------------------
The closure of the following set of attributes:
snum
Answer >
snum sname status city
-----------------------------------------------------------------
The dependency basis of the following set of attributes:
snum
Answer >
> snum
> sname
> status
> city
> pnum pname weight color qty
-----------------------------------------------------------------
```

Figure 6-2 shows the result of the above query. Elements belonging to the same subset of the dependency basis of 'snum' are marked with the same label 'Si'.

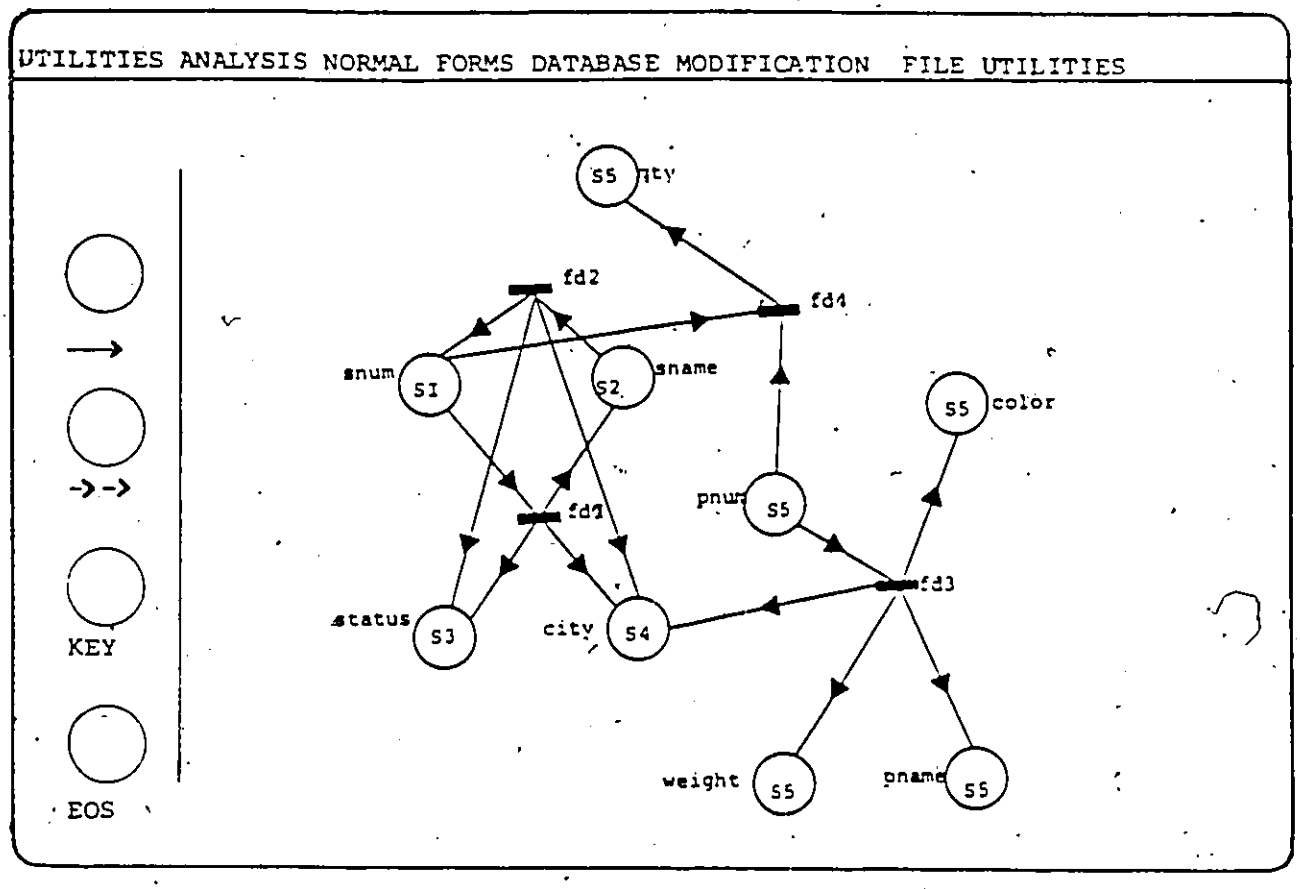


Figure 6-2. Graphical result displayed on the screen.

```
---------------------------------------------------------------
The closure of the following set of attributes:
pnum
Answer >
city pnum pname weight color
---------------------------------------------------------------
The dependency basis of the following set of attributes:
pnum
Answer >
> pnum
> city
> pname
> weight
> color
> snum sname status qtyor qty
---------------------------------------------------------------
Testing whether (snum,sname) is a key of the set of attributes
  (snum,sname,status,city,qty)
No, it is not.
---------------------------------------------------------------
The Key of the relation scheme:
 snum,sname,status,city
 Answer >
  snum
  sname
---------------------------------------------------------------
The Key of the relation scheme:
 snum,pnum,qty,color
 Answer >
  snum pnum
---------------------------------------------------------------
The Key of the relation scheme:
 snum,sname,pnum,pname,color,qty,weight,city,status
 Answer >
  snum pnum
---------------------------------------------------------------
The closure of the following set of attributes:
snum,pnum
Answer >
snum sname status city pnum pname weight color qty
---------------------------------------------------------------
The dependency basis of the following set of attributes:
snum,pnum
Answer >
 > snum
 > pnum
 > sname
 > status
 > city
 > pname
 > weight
 > color
 > qty
---------------------------------------------------------------
```

Following is a record of the normal forms generation and acyclicity test for example 6.1.

```
-------------------------------------------------------------------
*************** NORMAL FORMS AND ACYCLICITY TEST *****************
-------------------------------------------------------------------

****** Synthesized 3NF relation schemes. ******
-------------------------------------------------------------------
 snum sname status city
 city pnum pname weight color
 snum pnum qty
-------------------------------------------------------------------
****** Generated BCNF relation schemes. ******
-------------------------------------------------------------------
snum sname status city
Key :
snum
--------
city pnum pname weight color
Key :
pnum
-------
snum pnum qty
Key :
snum pnum
-------------------------------------------------------------------
****** Generated 4NF relation schemes. ******
-------------------------------------------------------------------
snum sname
snum status
snum city
pnum pname
pnum weight
snum pnum qty
pnum color
-------------------------------------------------------------------
Testing whether the following scheme is in 4NF.
snum,sname,status,city
Yes it is.
-------------------------------------------------------------------
Testing whether the following scheme is in 4NF.
pnum,color,qty
No it is not.
-------------------------------------------------------------------
Testing whether the following scheme is in 4NF.
pnum,pname,color,weight,qty
No it is not.
-------------------------------------------------------------------
```

```
------------------------------------------------------------------
****** ACYCLICITY OF DATABASE SCHEME ******
------------------------------------------------------------------
The database scheme is:
 snum,sname,status
 sname,status,city
 status,city,pnum

The database scheme is a-acyclic. Graham reduction succeeded.

------------------------------------------------------------------
****** ACYCLICITY OF DATABASE SCHEME ******
------------------------------------------------------------------
The database scheme is:
 snum,sname,status
 sname,status,city
 status,pnum
 city,pnum

The database scheme is a-cyclic. Graham reduction failed.

------------------------------------------------------------------
--------------- END OF GLORD SESSION  ----------------------------
------------------------------------------------------------------
```

Example 6.2

Consider a relational database having a universe of atributes U and a set of data dependencies F.

U = { a, b, c, d }

F = { a --> b,
d --> c,
(ab, bc, dc) }.

After querying GLORD about some analysis problems, we added a multivalued dependency to F. Figure 6-3 shows the EPN representation after adding the MVD.

```
-------------------------------------------------------------
--------------- GLORD DESIGN SESSION ------------------------
-------------------------------------------------------------
Given : The database we design and test is: DBASE:EX2
-------------------------------------------------------------
 The universe of attributes of this database is:
 a b c d
-------------------------------------------------------------
 The data dependencies are:

 fd1 : a --> b
 fd2 : d --> c
  s1 : ( a b , b c , c d )

-------------------------------------------------------------
***************** ANALYSIS OF DATA DEPENDENCIES ***************
-------------------------------------------------------------

Testing whether the following data dependency is logically implied:
a ->-> c
Answer >>
Not logically implied.
-------------------------------------------------------------
Testing whether the following data dependency is logically implied:
a ->-> d
Answer >>
Not logically implied.
-------------------------------------------------------------
```

```
---------------------------------------------------------------
The closure of the following set of attributes:
a
Answer >>
a b
---------------------------------------------------------------
The dependency basis of the following set of attributes:
a
Answer >>
a
b
d c
---------------------------------------------------------------

  An MVD is added to the existing set of data dependencies.

---------------------------------------------------------------
 The universe of attributes of this database is:
 a b c d
---------------------------------------------------------------
 The data dependencies are:

 fd1 : a --> b
 fd2 : d --> c
 mvd : a ->-> c
  sl : ( a b , b c , c d )

---------------------------------------------------------------
```

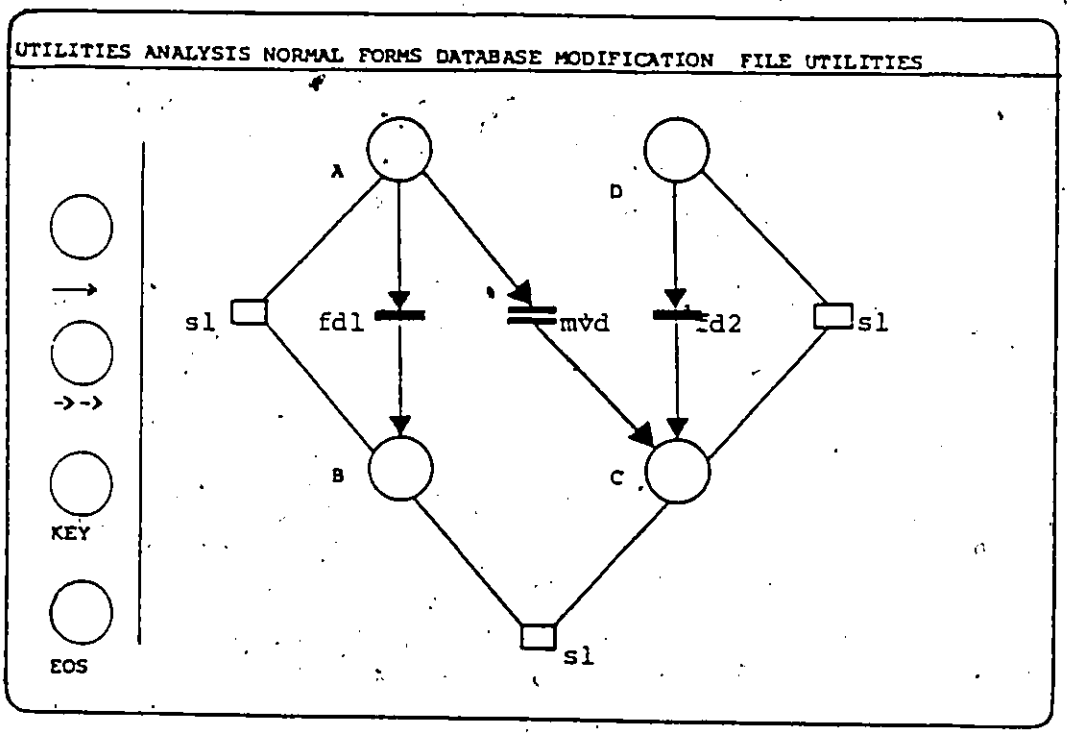


Figure 6-3. EPN representation of Example 6.2.

```
-----------------------------------------------------------------
The closure of the following set of attributes:
a
Answer >>
a b c
-----------------------------------------------------------------
The dependency basis of the following set of attributes:
a
Answer >>
a
b
c
d
-----------------------------------------------------------------
```

Following is a record of the normal forms generation and acyclicity test. The MVD added in the analysis session is excluded from the database.

```
-----------------------------------------------------------------
****** ACYCLICITY OF DATABASE SCHEME *******
-----------------------------------------------------------------
The database scheme is:
s1 : (a b , b c , c d)

The database scheme is alpha-acyclic. Graham reduction succeeded.
-----------------------------------------------------------------
Testing whether the following scheme is in 4NF:
a,b,c
No, it is not.
-----------------------------------------------------------------
****** Generated 4NF relation schemes ******
-----------------------------------------------------------------
a b
a d
c d
-----------------------------------------------------------------
Testing whether the following scheme is in 4NF:
a,c
Yes, it is.
-----------------------------------------------------------------
Testing whether the following scheme is in 4NF:
a,c
Yes, it is.
-----------------------------------------------------------------
-------------- END OF DATABASE DESIGN SESSION -------------------
-----------------------------------------------------------------
```

Example 6.3

Consider the following database:

U = { a b c d e }

F = { (a b , c d , b c e),
(a b , a c , a d e) }

Following is the printed record of an analysis, normal forms generation and acyclicity test session. An FD was added to F in the middle of the session.

```
-------------------------------------------------------------
---------------- GLORD DESIGN SESSION -----------------------
-------------------------------------------------------------

Given : The database we design and test is: DBASE:EX3
-------------------------------------------------------------
 The universe of attributes of this database is:
 a b c d e
-------------------------------------------------------------
 The data dependencies are:

 s1 : ( a b , b c e , c d )
 s2 : ( a b , a c , a d e ).

-------------------------------------------------------------
The closure of the following set of attributes:
b
Answer >>
b
-------------------------------------------------------------
The dependency basis of the following set of attributes:
b
Answer >>
b
a
c
e
d
-------------------------------------------------------------
```

```
------------------------------------------------------------------
Testing whether the following data dependency is logically implied:
b ->-> a c
Answer >>
Logically implied.
------------------------------------------------------------------
Testing whether: a b d is a key of: a b c d e .
No, it is not.
------------------------------------------------------------------

 The universe of attributes of this database is:
 a b c d e
------------------------------------------------------------------
 The data dependencies are:

 fd1 : a --> c
 s1  : ( a b , b c e , c d )
 s2  : ( a b , a c , a d e ).

------------------------------------------------------------------
The closure of the following set of attributes:
b
Answer >>
b c
------------------------------------------------------------------
The dependency basis of the following set of attributes:
b
Answer >>
b
a
c
d
e
------------------------------------------------------------------
The closure of the following set of attributes:
a
Answer >>
a c
------------------------------------------------------------------
The dependency basis of the following set of attributes:
a
Answer >>
a
c
d
b
e
```

```
------------------------------------------------------------
************ NORMAL FORMS AND ACYCLICITY TEST *******
------------------------------------------------------------
****** GENERATED   4NF relation schemes  ******
------------------------------------------------------------
a c
b d e
a b
------------------------------------------------------------
Testing whether the following scheme is in 4NF:
b,c,e
Yes, it is.
------------------------------------------------------------
Testing whether the following scheme is in 4NF:
a,b,c
No, its is not.
------------------------------------------------------------
******* ACYCLICITY OF DATABASE SCHEME ********
------------------------------------------------------------
The database scheme is:
 s1 : ( a b , b c e , c d )

The database scheme is alpha-acyclic. Graham reduction succeeded.
------------------------------------------------------------
The database scheme is:
 s1 : ( a b , b c e , c d )

The database scheme is alpha-acyclic. Graham reduction succeeded.
------------------------------------------------------------
------------- END OF DATABASE DESIGN SESSION --------
------------------------------------------------------------
```

6.1 SUMMARY AND CONCLUSION

The logical design of relational database schemes is mainly concerned with the following problems: (i) Given a set of attributes and a set of data constraints, form the database schemes so that they satisfy the various normal forms with respect to these constraints. (ii) Given a database scheme and a set of data dependencies, test whether the scheme satisfies other constraints or not. The former problem requires the generation, by decomposition or synthesis, of normalized relation schemes. The latter requires the solution of the membership or implication-related problems.

We have introduced a graphical model by extending the ordinary Petri-net. In comparison with other existing models, our model has the following advantages: (i) It can handle three kinds of transitions, namely FD-, MVD- and R-transitions for the representation of FDs, MVDs and relation schemes, respectively. (ii) It is based on a strong mathematical foundation for handling FD-transitions and R-transitions. (iii) It has unique representations for three types of data dependencies. (iv) It is clear and appealing to viewers' intuition. (v) It is easy for computerization. These advantages ensure that this model is superior to the other existing models.

An interactive graphic system GLORD based on this model has been implemented. It consists of three subsystems: i) The graphic support subsystem providing the interactive user interface. ii) Design and test subsystem. iii) Help subsystem providing help messages for the user. One limitation of GLORD is that it can accept a maximum of 32 attributes in a universe.

One technical difficulty encountered while developing GLORD was due to the fact that the Macintosh Pascal interpreter does not accept large programs.

Our experience with GLORD demonstrates its capability to test and design relational databases in a user-friendly graphical environment.

A related area for investigation is the development of graphical query languages for database manipulation. In fact, our graphical system is being expanded to include such a language under another project.

Appendix A

A USER'S GUIDE TO GLORD

A.1 INTRODUCTION

The Graphic system for LOgical Relational database Design (GLORD) is a menu-driven graphic system for testing the logical design of relational databases. This appendix describes GLORD at the operational level. Section A.2 explains how to enter GLORD. Section A.3 describes the functions of GLORD. Section A.4 explains how to quit from GLORD.

A.2 ENTERING GLORD

To enter GLORD, we first have to open the GLORD program by selecting entry OPEN from the Macintosh Pascal menu. We then select entry GO from the menu RUN. GLORD program will then be compiled and a welcome message appears on the screen.(Figure A-1).

The system will then prompt for the name of the database you want to work on. You may select an existing database (if any) or start with a new one by clicking CANCEL (Figure A-2).

If an existing database is selected, GLORD displays its menu headings. You may select any entry from any menu heading.

If a new database is started, GLORD displays all menu headings. In order to start building it, you should select DISPLAY DATABASE from menu UTILITIES. A blank screen with the menu headings is displayed, and the system is ready now to accept any selection from menu DATABASE MODIFICATION.

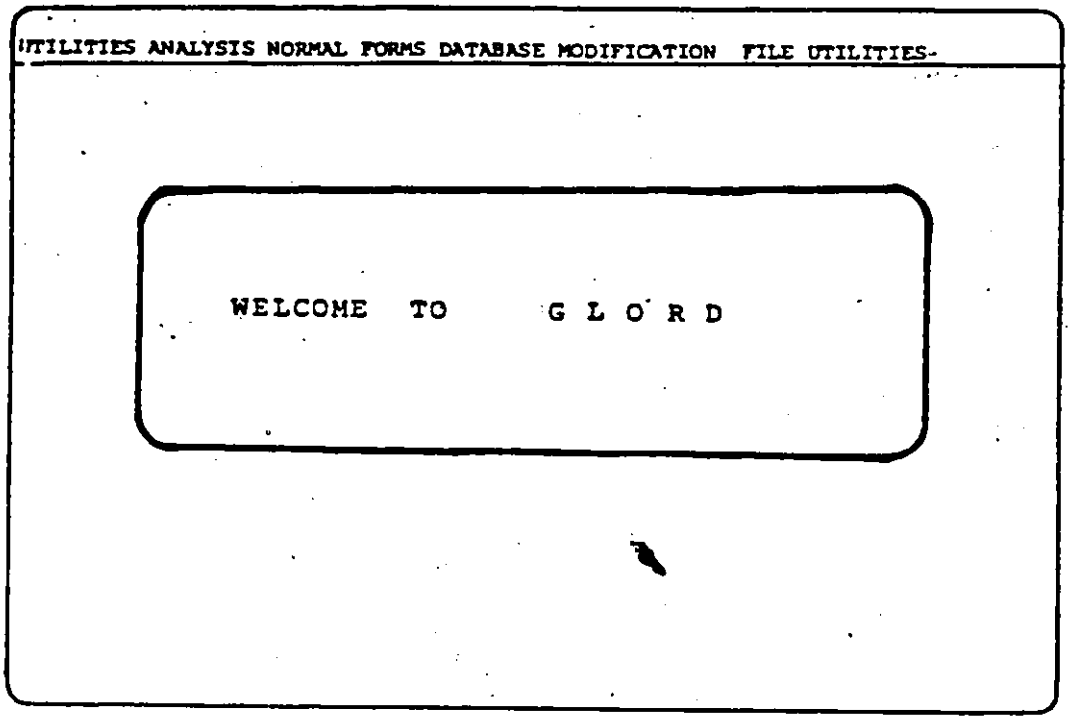


Figure A-1. Welcome message of GLORD.

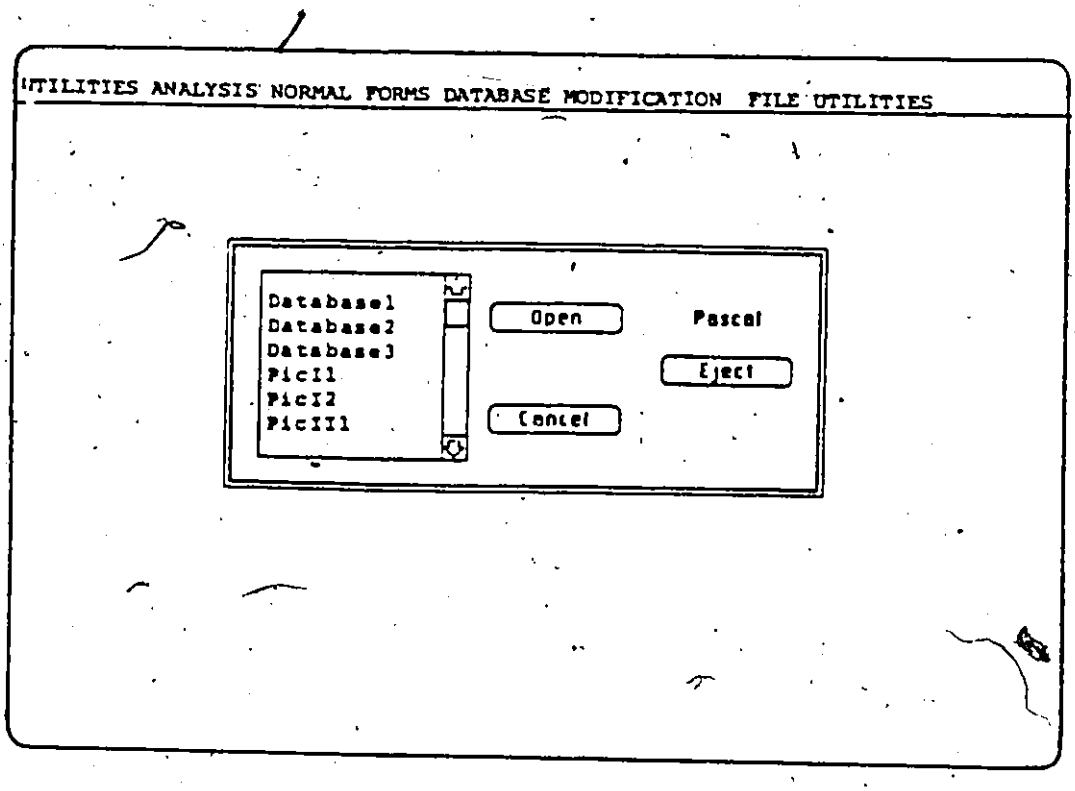


Figure A-2. Selection of a database.

A.3 FUNCTIONS, USER/SYSTEM INTERACTIONS AND MESSAGES

This section describes the user/system interaction (if any) and associated help message for each entry of the menus. The help message for an entry will appear on the screen if the entry HELP is selected. Figure A-3 shows the initial screen when GLORD is started.

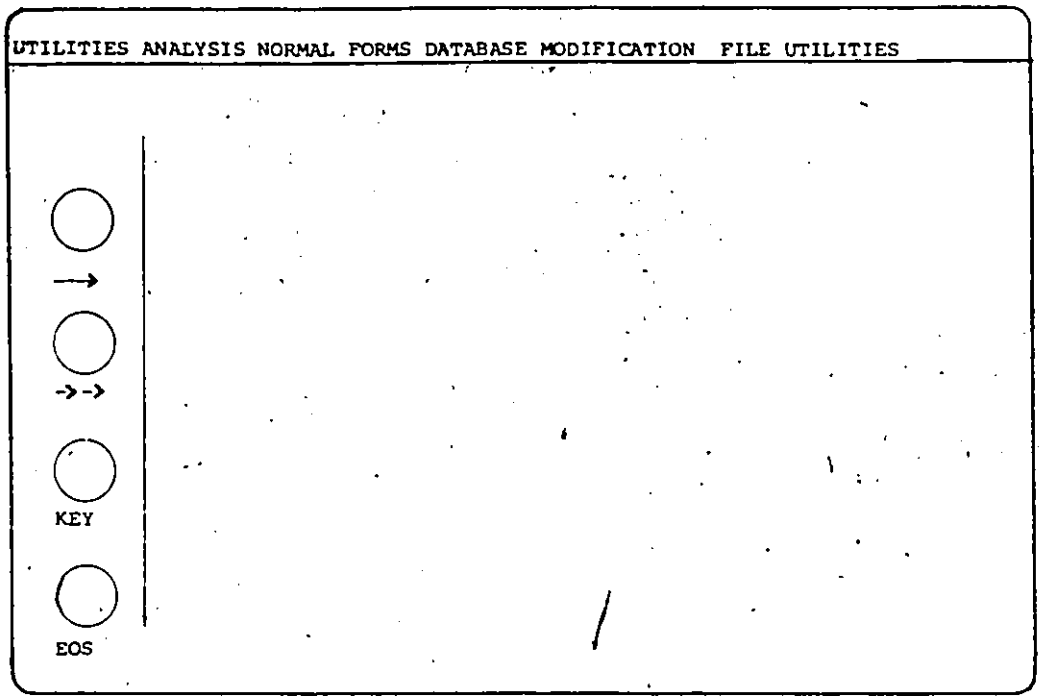


Figure A-3. Initial screen of GLORD.

A.3.1 Menu heading: UTILITIES

Menu entries:

1. DISPLAY DATABASE
2. CLEAR
3. PRINT DATABASE
4. NAME
5. QUIT
6. HELP

- DISPLAY DATABASE

The EPN representation of the database currently under test is displayed. This includes the names of all the attributes and data dependencies.

No query or answer is expected from the system.

Help message: Help on DISPLAY DATABASE. Display the database under test. The EPN representation of attributes and data dependencies will be displayed.

- CLEAR

The screen will be cleared. This entry is usually selected when the screen contains used data and when a new database is to be created.

No query or answer is expected from the system.

Help message: Help on CLEAR. Screen is cleared.

- PRINT DATABASE

This selection will cause all the current information about the database (data dependencies and attributes) to be printed.

The system will respond as follows:

```
-------------------------------------------------
GIVEN: The database we design and test is: dbase.ex1
-------------------------------------------------
 The universe of this database is:

  A B C D ..
-------------------------------------------------
 The data dependencies are:
  fd1: X --> Y
  ....
-------------------------------------------------
```

where A, B, C, ... can be the names of the attributes and X, Y are lists of the names of attributes.

Help message: Help on PRINT DATABASE. Print the database under test. The universe and the data dependencies will be printed.

- NAME

This selection permits the user to name any newly created data dependency or attribute in the EPN representation of a database. It can also be used to rename an existing data dependencies or attribute.

After selecting NAME, the user should click at the required element to name (or rename). The system will prompt: "ENTER NAME:"

The system expects a string of 16 characters maximum.

Help message: Help on NAME. To name or rename an FD-type, an MVD-type , a JD-type of transitions or an attribute in the database under test.

- QUIT

At the end of a design session, the user can choose the entry QUIT. The user will be asked if he wants to save the modifications and test a new database.

First, the system will prompt:

Do you want to save the database changes (if any) ? (y/n)

Then it prompts:

Do you want to quit the system ? (y/n).

Help message: Help on QUIT. To quit the database under test, switch to another database or leave the system.

- HELP

This entry is selected when the user needs information about the function performed by an entry in the menu UTILITIES. A help message for a particular entry (function) will be displayed.

Help message: To get detailed information for each of the entries in the menu UTILITIES, please enter: 1,2,3,4 or 5.

A.3.2 Menu heading: ANALYSIS

Menu entries:

1. IMPLICATION
2. CLOSURE
3. KEY
4. HELP

- IMPLICATION

In the analysis aspect of a logical design, this problem is known as the "membership" or "logical implication" problem. Given the set DC (data constraints) containing the FDs, MVDs and JDs, we should be able to test whether an FD or an MVD is logically implied by DC. This test is denoted by: DC |= FD or DC |= MVD. In practice, the implementation of the system does not allow the test of join dependencies: DC |= JD. This test has been proven to be NP-Hard.

The system expects the user to enter graphically a data dependency using the mouse. If an FD (or MVD) is to be tested, the user should click at the places representing the attributes at the left-hand side, click at --> (or ->->) on the left margin of the screen, click at the places representing the attributes at the right-hand side and then click at EOS.

The system responds by either 'yes' or 'no' on both the screen and the printer.

Help message: Help on IMPLICATION. Test if an FD or an MVD is logically implied by given data dependencies.

- CLOSURE

The selection CLOSURE will perform the computation of both the closure and the dependency basis of a set of attributes.

The CLOSURE finding problem is one of the basic problems that are solved in this system. The dependency basis of a set of attributes allows us to test the membership problem for MVDs and to develop relation schemes satisfying the fourth normal form.

The system expects the user to click at the places representing the given set of attributes and then click at EOS.

GLORD displays the EPN representation. The places belonging to the same subset of the dependency basis are marked with the same index 'Si', i= 1,2, ...

The printed results are:

```
The closure of the following set of attributes:

A B C D
Answer >
A B C D E G H  ...
-----------------------------------------------
The dependency basis is:
Answer >>
A
B
C D
....
```

Help message: Help on CLOSURE. To find the closure and the dependency basis of a set of attributes. Attributes are entered by clicking at the attributes and then clicking at EOS.

• KEY

This selection finds either the key of a given set of attributes or it tests if a given set of attributes is a key of another set of attributes.

If the user wants to test whether a set of attributes K is a key for another set of attributes S, the system expects him/her to click at the places representing S, click at KEY, click at the places representing K, then click at EOS.

The answer expected from the system is either 'YES' or 'NO' next to KEY. The printed result is: (e.g.)

```
Testing whether (A,C) is a key of the scheme
  (A,B,C,D,E):

Answer > Yes.  (or No.)
```

If the user wants to find a key for a set of attributes, the system expects him/her to click at the places representing the attributes, then click at EOS.

The attributes of the key will be labeled with k. The printed result is:

A key for the following set of atributes: ..... .

Help message: Help on KEY. Test whether a set of attributes is a key of another set of attributes or find a key for a relation scheme (a set of attributes).

• HELP

This entry is selected when the user needs information about the function performed by an entry in the menu ANALYSIS.

A help message for a particular entry (function) will be displayed.

Help message: To get detailed information for each of the entries in the menu ANALYSIS, please enter: 1,2 or 3.

A.3.3 Menu heading: NORMAL FORMS

Menu entries:

1. SYNTHESIZE 3NF
2. TEST 3NF
3. GENERATE BCNF
4. GENERATE 4NF
5. TEST 4NF
6. TEST ACYCLICITY
7. HELP

Entries TEST 3NF and TEST 4NF can accept input either graphically or manually.

- SYNTHESIZE 3NF

This selection will generate, by synthesis, relation schemes in third normal form.

The system will print all the synthesized schemes which are in third normal form:

```
-----------------------------------------------
****** Synthesized 3NF relation schemes ******
-----------------------------------------------
A,B,C,D   .Key-A
....
....
```

Help message: Help on SYNTHESIZE 3NF. To synthesize relation schemes in third normal form.

- TEST 3NF

This selection will test whether a given relation scheme is in third normal form.

The system will prompt for a relation scheme and expects a sequence of attributes separated by commas.

Enter relation scheme:
==>>
The answer will be either 'Yes, it is' or 'No, it is not'.

Graphical input:

The system expects the user to click at all places representing the attributes of the scheme to test and then click at EOS.

Help message: Help on TEST 3NF. To test whether a given relation scheme is in third normal form.

- GENERATE BCNF

This selection will generate relation schemes in Boyce-Codd normal form. A decomposition algorithm is used to decompose the Universe into relation schemes in Boyce-Codd normal form.

The output of the system is:

```
-----------------------------------------------
****** Generated BCNF relation schemes ******
-----------------------------------------------

A,B,C,D
Key : A
-----
....
....
```

Help message: Help on GENERATE BCNF. To generate relation schemes in Boyce-Codd normal form.

- GENERATE 4NF

This selection will decompose the universe of attributes into relation schemes in fourth normal form.

The system will print all the relation schemes.

```
-----------------------------------------------
****** Generated    4NF    relation schemes ******
-----------------------------------------------

A,B
A,C,D
....
```

Help message: Help on GENERATE 4NF. To generate relation schemes in Fourth normal form.

- TEST 4NF

This selection tests whether a given relation scheme is in fourth normal form.

The system will prompt for a relation scheme and expects a sequence of attributes separated by commas.

Enter relation scheme:
==>>

The answer will be either 'Yes, it is' or 'No, it is not'.

Graphical input:

The system expects the user to click at all places representing the attributes of the scheme to test and then click at EOS.

Help message: Help on TEST 4NF. To test whether a given relation scheme is in fourth normal form.

- TEST ACYCLICITY

This selection will test whether a given database scheme is alpha-acyclic or not.

The system will prompt for relation schemes until a period '.' is entered.

The database scheme is: .....

The answer will be one of the following:
The database scheme is alpha-acyclic. Graham reduction succeeded.
or The database scheme is alpha-cyclic. Graham reduction failed.

Help message: Help on TEST ACYCLICITY. Test whether the given database scheme is a-acyclic. The Graham reduction algorithm is used:

• HELP

This entry is selected when the user needs information about the function performed by an entry in the menu NORMAL FORM. A help message for a particular entry (function) will be displayed.

Help message: To get detailed information for each of the entries in the menu NORMAL FORMS, please enter 1,2,3,4,5 or 6.

A.3.4 Menu heading: DATABASE MODIFICATION

Menu entries:

1. ADD ARC
2. ADD FD
3. ADD MVD
4. ADD SCHEME
5. ADD ATTRIBUTE
6. DELETE ARC
7. HELP

A.3.4.1 UTILITIES for the graphic interface of the system

Entries ADD ARC, ADD FD, ADD MVD, ADD JD, ADD ATT and DELETE ARC

During the test and design processes, we may have to modify the database design and test its impact on the existing design.

Attributes, FDs, MVDs, relation schemes and arcs can be added or deleted graphically to the existing database, and new testing session can be started. (To start with a new database, see ACCEPT MODIFICATION in menu FILE UTILITIES). In addition, existing arcs can be deleted from the database, this imply the deletion of some existing data dependencies.

- HELP

This entry is selected when the user needs information about the menu DATABASE MODIFICATION. A help message for a particular entry (function) will be displayed.

Help message: To get detailed information for each of the entries in the menu DATABASE MODIFICATION, please enter 1,2, 3,4,5 or 6.

A.3.5 Menu heading: FILE UTILITIES

Menu Entries:

1. CREATE DATABASE
2. SAVE PICTURE
3. ACCEPT MODIFICATION
4. RECORD ON
5. RECORD OFF
6. PRINT REC.
7. HELP

- CREATE DATABASE

This selection allows the user to create a new database design.

The system will prompt for the database name and then for the maximum number of FDs, MVDs and JDs:

```
Enter Maximum number of FDs:
==>>

Enter Maximum number of MVDs:
==>>

Enter Maximum number of JDs:
==>>
```

The total of the three entered numbers should be less than 33.

Help message: Help on CREATE. To create a new database.

- SAVE PICTURE

This selection allows the user to save the EPN representation of the database displayed on the screen.

The prompt from the system is:

```
Enter file name for saved picture:
==>>
```

This file can be printed using MacPaint, the Macintosh drawing tool.

Help message: Help on SAVE PICTURE. To save actual drawing on screen. You will be prompted for a file name to contain picture.

- ACCEPT MODIFICATION

This selection will save any modification to the database made by an entry from menu DATABASE MODIFICATION.

The system query is:

```
Do you want to save changes made so far? (y/n).
==>>
```

Help message: Help on ACCEPT MODIFICATION. To accept intermediate database modifications. You will be prompted for whether you want to save or not.

- RECORD ON

This selection will turn on the recording feature of GLORD. All user/system interactions will be recorded in a separate file.

The system will display:

Ack: Your recording of session is ON.

If recording is already on, the system display a warning:

WARNING:
Your recording is already ON.

Help message: Help on RECORD ON. Recording is a utility that records all user/system interactions during the design testing session. This selection will turn it on.

- RECORD OFF

This selection will turn off the recording feature of GLORD.

Once this entry is selected, the system will display:

Ack: Your recording of session is OFF.

If recording is already on, the system display a warning:

WARNING:
Your recording is already OFF.

Help message: Help on RECORD OFF. Recording is a utility that records all user/system interactions during the design testing session. This selection will turn it off.

- PRINT REC.

The content of the recording file generated by the recording feature is printed.

Help message: Help on PRINT REC. To get a hard-copy of the recorded user/system interactions.

- HELP

This entry is selected when the user needs information about the function performed by an entry in the menu DATABASE MODIFICATION. A help message for a particular entry (function) will be displayed.

Help message: To get detailed information for each of the entries in menu FILE UTILITIES, please enter 1,2,3,4,5 or 6.

A.4 EXITING GLORD

The selection of entry QUIT from the menu UTILITIES will allow you to either quit the system or start as if you are entering the system. In case of quitting the system, an end-of-session message is displayed. If you choose not to quit the system, you will be able to test or to create a new database.

Appendix B

MESSAGES IN GLORD

```
**********************************************************
***  MESSAGE FILE FOR HELP, ERROR, QUERY      ***
***      AND PROMPT MESSAGES.                 ***
**********************************************************


----------------------------
 ***   MSG:        1      *** |
----------------------------
     To get detailed information on each of the entries in
     UTILITIES menu, please enter 1, 2 ,3,4 or 5
      ==>>
----------------------------
| ***   MSG:        4      *** |
----------------------------
     Help on DISPLAY DATABASE. Display the database under test.
     The EPN representation of attributes and data dependencies
     will be displayed
----------------------------
| ***   MSG:        7      *** |
----------------------------
     Help on CLEAR. Clear Screen.


----------------------------
| ***   MSG:        10     *** |
----------------------------
     Help on KEY. Test whether a set of attributes is
     a Key of another set of attributes or find a key for a relation
     scheme. Click at places for att. of scheme, then click at att. for key.
----------------------------
| ***   MSG:        12     *** |
----------------------------
     To get detailed information on each of the entries in
     menu ANALYSIS, please enter 1,2 or 3.
      ==>>
----------------------------
| ***   MSG:        16     *** |
----------------------------
     Help on CLOSURE. To find the closure and the dependency basis
     of a set of attributes.Click at places corresponding to the set,
     then click at place EOS,
----------------------------
| ***   MSG:        19     *** |
----------------------------
     Help on DEPENDENCY BASIS. To find the dependency basis
     of a set of attributes. The attributes should be entered as follows:
     ==>>  student,address,courses
----------------------------
| ***   MSG:        22     *** |
----------------------------
     Help on IMPLICATION. Find if an Mvd or an Fd is logically
     implied by data dependencies. Click at places for att. at l.h.s. then
     click --> or ->-> followed by places for att. at r.h.s., then click EOS
----------------------------
 ***   MSG:        25     *** |
----------------------------
     To get detailed information on each of the entries in
     menu DATABASE MODIFICATION, please enter 1,2,3,4 or 5.
      ==>>
```

```
*** MSG: 28 ***
```

Help on PRINT DATABASE. Print the database under test.
The universe and the data dependencies will be printed.

```
*** MSG: 31 ***
```

Help on TEST. Print the set of MVDs that is equivalent
to the given data dependencies.

```
*** MSG: 34 ***
```

Help on QUIT. To quit the database under test : switch to another
database or leave the system.

```
*** MSG: 37 ***
```

To get detailed information on each of the entries in
menu FILE UTILITIES, please enter 1,2 or 3.
==>>

```
*** MSG: 40 ***
```

RECORDING is a utility used to all interactions User/system
for the design testing session. The default is OFF
CHOOSE ON to turn it on.

```
*** MSG: 43 ***
```

Select 1. or 2. Enter (1/2).
1. Testing whether a given set of attributes is a Key.
2. Finding a Key for a set of attributes.

```
*** MSG: 46 ***
```

Enter file name for saved picture.
==>>

```
*** MSG: 49 ***
```

Enter data dependency please.
==>>

```
*** MSG: 52 ***
```

Help on CREATE. To create a new database.
You will be prompted for more information.

```
*** MSG: 55 ***
```

Enter a set of attributes please.
==>>

```
----------------------------
*** MSG:      53    *** !
----------------------------
     HELP ON HELP MOD.

----------------------------
*** MSG:      61    *** !
----------------------------
     Help on PRINT REC. To print database design queries results
     recorded in recording file.

----------------------------
*** MSG:      64    *** !
----------------------------
     ACK: Your recording of session is ON

----------------------------
*** MSG:      67    *** !
----------------------------
     The Key of the relation scheme:

----------------------------
*** MSG:      70    *** !
----------------------------
     Enter the set of attributes or Scheme
      ==>>

----------------------------
*** MSG:      72    *** !
----------------------------
     Enter Key for test:
      ==>>

----------------------------
*** MSG:      76    *** !
----------------------------
     Help on ACCEPT. To accept intermediate database modifications.
     You will be prompted for whether you want to save or not.

----------------------------
*** MSG:      79    *** !
----------------------------
     Do you want to save the database changes (if any)? (y/n)
      ==>>

----------------------------
*** MSG:      82    *** !
----------------------------
     Do you want to quit the system? (y/n)
      ==>>

----------------------------
*** MSG:      85    *** !
----------------------------
     The closure of the following set of attributes:
```

```
-----------------------------
! ***   MSG:        88    *** !
-----------------------------
     The dependency basis :


-----------------------------
! ***   MSG:        91    *** !
-----------------------------
     Testing whether the following data dependency is logically implied


-----------------------------
! ***   MSG:        94    *** !
-----------------------------
     Testing whether the given set of attributes is a Key.


-----------------------------
! ***   MSG:        97    *** !
-----------------------------
     Do you want to save changes made so far? (y/n).
      ==>>

-----------------------------
! ***   MSG:       100    *** !
-----------------------------
     Error on help menu UTILITIES :
     You have entered an invalid choice of help item
     Try help on menu UTILITIES  again
-----------------------------
  ***   MSG:       103    *** !
-----------------------------
     Error on help menu ANALYSIS :
      You have entered an invalid choice of help item
     Try help on menu ANALYSIS  again
-----------------------------
! ***   MSG:       106    *** !
-----------------------------
     Error on help menu DATABASE MODIFICATION:
      You have entered an invalid choice of help item
     Try help on menu DATABASE MODIFICATION again
-----------------------------
! ***   MSG:       109    *** !
-----------------------------
     Error on Help menu FILE UTILITIES :
      You have entered an invalid choice of help item
     Try Help on menu FILE UTILITIES  again.
-----------------------------
! ***   MSG:       112    *** !
-----------------------------
     ERROR:
     You Entered a QUERY Wrongly.
     SELECT Help on these item .
-----------------------------
  ***   MSG:       115    *** !
-----------------------------
     ERROR
     ONE or MORE Attributes are not in UNIVERSE
```

```
---------------------------
*** MSG:      88     ***
---------------------------
     The dependency basis :


---------------------------
*** MSG:      91     ***
---------------------------
     Testing whether the following data dependency is logically implied


---------------------------
*** MSG:      94     ***
---------------------------
     Testing whether the given set of attributes is a Key.


---------------------------
*** MSG:      97     ***
---------------------------
     Do you want to save changes made so far? (y/n).
     ==>>

---------------------------
*** MSG:      100    ***
---------------------------
     Error on help menu UTILITIES :
     You have entered an invalid choice of help item
     Try help on menu UTILITIES  again
---------------------------
*** MSG:      103    ***
---------------------------
     Error on help menu ANALYSIS :
     You have entered an invalid choice of help item
     Try help on menu ANALYSIS  again
---------------------------
*** MSG:      106    ***
---------------------------
     Error on help menu DATABASE MODIFICATION:
     You have entered an invalid choice of help item
     Try help on menu DATABASE MODIFICATION again
---------------------------
*** MSG:      109    ***
---------------------------
     Error on Help menu FILE UTILITIES :
     You have entered an invalid choice of help item
     Try Help on menu FILE UTILITIES  again.
---------------------------
*** MSG:      112    ***
---------------------------
     ERROR:
     You Entered a QUERY Wrongly.
     SELECT Help on these item .
---------------------------
*** MSG:      115    ***
---------------------------
     ERROR
     ONE or MORE Attributes are not in UNIVERSE
```

```
------------------------------
| ***   MSG:          118      *** |
------------------------------
     WARNING:
     Your RECORDING is already ON.

------------------------------
| ***   MSG:          121      *** |
------------------------------
     WARNING:
     Your RECORDING is already OFF.

------------------------------
| ***   MSG:          124      *** |
------------------------------
     WARNING
     No more RECORDING will be done. You have exceeded
     allowed limit. Try Help on recording
------------------------------
| ***   MSG:          127      *** |
------------------------------
     Help on ADD ATT. To add a new attribute to the database Universe.


------------------------------
| ***   MSG:          130      *** |
------------------------------
     Help on ADD ARC. To add arcs joining places(att) and
     transitions(DBs).

------------------------------
| ***   MSG:          133      *** |
------------------------------
     Help on ADD FD. To add FD-type of transition to the Graphic .
     Adding an FD to the database under test.

------------------------------
| ***   MSG:          136      *** |
------------------------------
     Help on ADD MVD. To add MVD-type of transition to the Graphic.
     Adding an MVD to the database under test.

------------------------------
| ***   MSG:          139      *** |
------------------------------
     Help on ADD JD. To add JD-type of transition to the Graphic.
     Adding a JD to the database under test.

------------------------------
| ***   MSG:          142      *** |
------------------------------
     Help on DELETE ARC. To delete arcs joining places and transitions.


------------------------------
| ***   MSG:          145      *** |
------------------------------
     Help on NAME. To name or rename an FD-type,
     an MVD-type or a JD-type of transitions in the database under test.
```

*** MSG: 148 ***

Help on SAVE PICTURE. To save actual drawing on screen.
You will be prompted for a file name to contain picture.

*** MSG: 151 ***

Enter Maximum number of FDs:
==>>

*** MSG: 154 ***

Enter Maximum number of MVDs:
==>>

*** MSG: 157 ***

Enter Maximum number of JDs:
==>>

*** MSG: 160 ***

The universe of this database is:

*** MSG: 163 ***

Logically implied.

*** MSG: 166 ***

Not logically implied.

*** MSG: 169 ***

Yes, it is.

*** MSG: 172 ***

No, it is not.

*** MSG: 175 ***

Answer >

*** MSG: 178 ***

----------DATABASE DESIGN SESSION ----------

*** MSG: 181 ***

---------- END OF DATABASE DESIGN SESSION ----------

*** MSG: 184 ***

ENTER NAME:

*** MSG: 187 ***

The data dependencies defined on the database are:

*** MSG: 190 ***

*** MSG: 193 ***

The database scheme is:

*** MSG: 196 ***

ERROR. Data dependency entered wrongly.
For help choose help on item 2.

*** MSG: 199 ***

Testing whether the following scheme is in 4NF.

*** MSG: 202 ***

To get detailed information on each of the entries in
menu NORMAL FORMS . please enter 1,2,3,4, 5 or 6.
==>>

*** MSG: 205 ***

Error on Help menu NORMAL FORMS:
You have entered an invalid choice of Help item.
Try Help on menu NORMAL FORMS again.

*** MSG: 208 ***

Help on GEN. BCNF. To Generate all relation schemes that are in Boyce-Codd normal form.

*** MSG: 211 ***

Help on GEN. 4NF. To Generate all relation schemes that are in fourth normal form.

*** MSG: 214 ***

Help on SYNTHESIZE 3NF. To Synthesize third normal forms relation schemes.

*** MSG: 217 ***

Help on TEST 4NF. To test whether a given relation scheme is in fourth normal form.

*** MSG: 220 ***

****** ACYCLICITY OF DATABASE SCHEME ******

*** MSG: 223 ***

****** Generated BCNF relation schemes. ******

*** MSG: 226 ***

****** Generated 4NF relation schemes. ******

*** MSG: 229 ***

****** Synthesized 3NF relation schemes. ******

*** MSG: 232 ***

Select 1. or 2. Enter (1/2).
1. Testing whether a given set of attributes is a Key.
2. Finding a Key for a set of attributes.

*** MSG: 235 ***

Help on TEST 3NF. To test whether a given relation scheme is in third normal form.

```
-----------------------------
| ***  MSG:      238     *** |
-----------------------------
```

Testing whether the scheme is in 3 Normal form.

```
-----------------------------
| ***  MSG:      241     *** |
-----------------------------
```

The Key of the relation scheme:

```
-----------------------------
| ***  MSG:      244     *** |
-----------------------------
```

Help on ACYCLICITY. Test whether the given database is acyclic. The Graham reduction algorithm is applied.

```
-----------------------------
| ***  MSG:      247     *** |
-----------------------------
```

The database scheme is Acyclic. Graham reduction succeeded.

```
-----------------------------
| ***  MSG:      250     *** |
-----------------------------
```

The database scheme is Cyclic. Graham reduction failed.

Appendix C

DESCRIPTION OF FUNCTIONS AND PROCEDURES OF GLORD

In this appendix, we briebly describe the functions and procedures in the graphic support and design and test subsystems of GLORD.

C.1 GRAPHIC SUPPORT FUNCTIONS AND PROCEDURES

- procedure ARC (flag)

This procedure draws or removes an arc from the graphic. This depends on the value of boolean variable 'flag'.

- procedure DF / DM / DJ / DRP (p)

These procedures draw an FD, MVD, JD or attribute (place), respectively, at position 'p' on the user's screen.

- procedure DSPNM

This procedure displays the FD, MVD, JD and attribute names of the corresponding database.

- procedure DSPTK

This procedure displays the number of tokens at every attribute (place) of the corresponding database.

- procedure ERT / FILLARROW

These two procedures are used by procedure 'ARC' for the addition and removal of arcs from the corresponding database.

- procedure HANDLEMENU

This is the main procedure that drives GLORD selection menus. It is mainly a loop waiting for a user selection of an entry. The appropriate procedure is called to execute user's request.

- procedure INIT

This procedure sets up the initial selection menus on the user's screen.

- procedure M (i: integer)

This procedure displays or prints help, query or answer message numbered 'i' in the message file opened by GLORD.

- procedure NPA

This procedure is used to allow the user to name or rename an FD, MVD, JD or attribute when requested.

- procedure NWN

This procedure allows the user to create a new database. It prompts for information about the number of FDs, MVDs and JDs.

- procedure PDB

This procedure prints the current database. The universe of attributes and all data dependencies are printed.

- procedure PM (p, m)

This procedure is called by 'DSPNM' to put the name 'm' of an FD, MVD, JD and attribute at position 'p' on the user's screen.

- procedure REDRAW

This procedure displays the attributes (places) and transitions (FDs, MVDs and JDs) of a database on the user's screen. The resulting graphic is the Extended Petri Net (EPN) representation of the database.

• procedure SCROUT

This procedure displays messages on the middle of the screen. These messages are WELCOME and END OF SESSION messages.

• procedure SDR

This procedure prepares the user's screen for drawing. It calls a Macintosh system procedure.

• procedure SX

This procedure initializes a screen area for displaying messages (queries, help or answers). It calls a Macintosh system procedure.

• function TESTMOUSE (bar: boolean) : boolean

This function tests for any event on the selection menu. It returns a boolean value.

C.2 DESIGN AND TEST FUNCTIONS AND PROCEDURES

• function CL (a: longint ; i: longint) : longint

This function computes the closure of the set of attributes 'a'. It uses different algorithms depending on the value of 'i'.

- procedure DEPB (cv: longint , x: longint)

This procedure computes the dependency basis of the set of attributes 'cv'. The elements of the dependency basis are stored in the global variable 'dt'.

- procedure DVD (sc: longint, rsc: longint)

This procedure decomposes (divides) the relation scheme 'sc' into two sets 'rsc' and 'sc'. If not decomposable, 'rsc' will be equal to 'sc'.

- procedure FINDKEY (a: longint, b: longint)

This procedure finds a key for the relation scheme 'a'. The key will be returned in 'b'.

- procedure GENBCNF

This procedure generates relation schemes in BCNF. It uses a decomposition algorithm.

- procedure GEN3NF

This procedure generates, by synthesis, relation schemes in 3NF. The Bernstein synthesis algorithm is used.

- procedure GR

This procedure applies the Graham reduction algorithm on the given relation schemes. These are represented by a Graham matrix.

- procedure IMPL

This procedure tests whether a given data dependency (FD or MVD) is logically implied by the existing data dependencies. It uses the global table 'dt'.

- function KEY (a: longint , b: longint) : boolean

This function tests whether the set of attributes 'a' is a key for the relation scheme 'b'.

- function TRIVIAL (a: longint, b: longint, c: longint) : longint

This function checks if the MVD 'a' ->-> 'b' is trivial in relation scheme 'c'.

- procedure TR

This procedure transforms the data dependencies to MVDs. These MVDs are stored in tables (ip and op).

- function TST4NF

This function prompts for a relation scheme and tests whether it is in 4NF.

C.3 LIST AND BIT MANIPULATION FUNCTIONS AND PROCEDURES

- procedure AtoB (att: str80 , cv: longint)

This procedure converts the set of attributes 'att' to the long integer 'cv'. Position 'i' in 'cv' is set to '1' if the attribute at position 'i' is contained in 'att'.

- procedure BtoA (cv: longint , att: str80)

This procedure converts the longint 'cv' to a string 'att'. The string will be either printed or displayed on the user's screen.

- function BT (bts: longint , n: integer) : boolean

This function tests the value of the bit at position 'n' of the long integer 'bts'.

- function CMP (i: longint) : longint

This function complements the list of attributes 'i' with the universe of attributes.

- procedure CNCT (i: longint , j: longint)

This procedure concatenates two lists of attributes 'i' and 'j'. The resulting list is stored in 'i'.

- function FINDI (i : integer) : str10

This function returns the attribute at position 'i' in the universe of attributes.

- procedure FIND (at: str10 , pt: integer)

This procedure finds the position 'pt' of the attribute 'at' in the universe of attributes.

- function ICL (i: integer , j: integer) : boolean

This function tests if column 'i' is contained in column 'j' of the graham matrix.

- function INCL (i: longint , j: longint) : boolean

This function returns 'true' if list 'j' is included in list 'i'.

- function IR (i: longint , j: longint) : longint

This function returns the intersection list between the lists 'i' and 'j'.

- procedure MINUS (a: longint , b: longint, c: longint)

This procedure computes the complement of list 'b' in list 'a'. The result is returned in list 'c'.

- procedure SB (long: longint, pos: integer, bt: boolean)

This procedure sets bit at position 'pos' in 'long' to '1' if 'bt' is 'true'.

- procedure UNION (i: longint , j: longint , k: longint)

This procedure will compute the union of lists 'i' and 'j'. The result will be stored in list 'k'.

- function UN (tp: longint) : boolean

This function tests if the list 'tp' is a union of lists in the dependency basis table (dt).

```
program GLORD1;
 uses
  quickdraw1, quickdraw2;
 const
  TT = true;
  FF = false;
  mx = 32;
 type
  str10 = string[10];
  str256 = string[80];
  w = INTEGER;
  bl = boolean;
  q = longint;
  Ptr = ^q;
  Handle = ^Ptr;
  WindowRecord = array[1..78] of w;
  WindowPtr = ^WindowRecord;
  str24 = string[24];
  str32 = string[32];
  evr = record
    wt : w;
    message : q;
    wn : q;
    wr : q;
    modifiers : w;
   end;
  mgs = record
    mg : array[1..200] of string[80];
   end;
  net = record
    id : w;
    p, t, m, j, c, mxp, mf, mm, mj, mxc : 0..64;
    na : array[-32..mx] of string[16];
    para, act : array[-32..mx] of str32;
    l : array[-32..mx] of point;
    tn : array[-32..mx, 1..4] of w;
    kn, ou : array[1..mx] of q;
   end;
 var
  done, ib, fir, nomore, sv, nt, ns : bl;
```

```
 ev : evr;
 pc : str24;
 r : rect;
 att, atl : str256;
 delay, n, rc, x, mvd, tsz, psz, a, b, ai, bj, tl : w;
 om, nm, cv, clos, cvl : q;
 op, ip, dt : array[1..20] of q;
 whichwindow : windowptr;
 z : net;
 c : point;
 pmp, dbfile : str24;
 dbvar : file of net;
 msgvar : file of mgs;
 mt : mgs;
 y : text;
 OldMenuBar, fm, am, sm, um : Handle;
procedure wy;
begin
 writeln(y);
end;
procedure sdr;
begin
 hideall;
 setrect(R, 0, 30, 527, 357);
 setdrawingrect(R);
 showdrawing;
end;
function testmouse (var bar : bl) : bl;
begin
 testmouse := ff;
 bar := ff;
 repeat
 until BinLineF($A970, $017F, @ev);
 if ev.wt = 1 then
  begin
   testmouse := tt;
   if WinLineF($A92C, ev.wr, @whichwindow) = 1 then
    bar := tt;
  end;
end;
function fn (xx, yy, frm, toi, siz : w;
```

```
        var n : w) : bl;
 var
  con : bl;
begin
 n := frm;
 con := tt;
 while (n <= toi) and con do
  if (sqr(xx - z.l[n].h) + sqr(yy - z.l[n].v) > siz * siz) then
   n := n + 1
  else
   con := ff;
 fn := not con;
end;
function fnt (xx, yy : w;
         var n : w) : w;
begin
 if fn(xx, yy, -z.p, -1, psz, n) then
  fnt := 1
 else if fn(xx, yy, 1, z.t, tsz, n) then
  fnt := 3
 else if fn(xx, yy, z.mf + 1, z.mf + z.m, tsz, n) then
  fnt := 7
 else if fn(xx, yy, z.mf + z.mm + 1, z.mf + z.mm + z.j, tsz, n)
      then
  fnt := 15
 else
  fnt := 0;
end;
procedure sx;
begin
 setrect(r, 0, 30, 521, 100);
 settextrect(r);
 showtext;
end;
procedure scr (m : str32);
begin
 sdr;
 fillrect(0, 0, 322, 512, Gray);
 setrect(r, 40, 150, 472, 230);
 fillroundrect(r, 20, 20, white);
 frameroundrect(r, 20, 20);
```

```
 textface([bold, outline]);
 moveto(150, 200);
 drawstring(m);
end;
procedure pm (p : point;
        m, con, ele : str24);
 var
  h, v, sz : w;
begin
 if ele = 'p' then
  sz := psz + 2
 else if ele = 't' then
  sz := tsz;
 h := p.h - stringwidth(m) div 2;
 if con = 'n' then
  v := p.v + sz + 8
 else if con = 'p' then
  v := p.v - sz
 else if con = 'a' then
  v := p.v - sz - 14;
 moveto(h, v);
 drawstring(m);
end;
procedure fa (f, t : point;
         radf, radt : w;
         drawarrow : bl);
 var
  x1, y1, ang : w;
  r : rect;
begin
 setrect(r, t.h - 12, t.v - 12, t.h + 12, t.v + 12);
 pttoangle(r, f, ang);
 y1 := round(radt * cos(ang * 0.0175));
 x1 := round(radt * sin(ang * 0.0175));
 f.v := f.v - round(radf * cos((ang - 180) * 0.0175));
 f.h := f.h + round(radf * sin((ang - 180) * 0.0175));
 moveto(f.h, f.v);
 lineto(t.h + x1, t.v - y1);
 setrect(r, t.h + 2 * x1 - 12, t.v - 2 * y1 - 12, t.h + 2 * x1 + 12,
      t.v - 2 * y1 + 12);
 if drawarrow then
```

```
  paintarc(r, ang - 15, 30);
end;
function prect (p : point;
        n : w) : rect;
 var
  r : rect;
begin
 if n = 1 then
  setrect(r, p.h - psz, p.v - psz, p.h + psz, p.v + psz)
 else if n = 2 then
  setrect(r, p.h - tsz, p.v - round(tsz * 0.4), p.h + tsz, p.v +
       round(tsz * 0.4))
 else if n = 3 then
  setrect(r, p.h - tsz, p.v - round(tsz * 0.1), p.h + tsz, p.v +
       round(tsz * 0.1))
 else if n = 4 then
  setrect(r, p.h - tsz, p.v - round(tsz * 0.8), p.h + tsz, p.v +
       round(tsz * 0.8));
 prect := r;
end;

function bt (bts, n : q) : bl;
begin
 if odd(bitshift(bts, 1 - n)) then
  bt := tt
 else
  bt := ff;
end;
procedure dsptk;
 var
  i : w;
begin
 textface([bold]);
 for i := 1 to z.p do
  begin
   moveto(z.l[-i].h - 6, z.l[-i].v + 4);
   if nt then
    begin
     att := concat('S', stringof(z.tn[-i, 1] : 1));
     drawstring(att);
    end
```

```
    else
     drawstring(stringof(z.tn[-i, 1] : 1));
   end;
  nt := ff;
  textmode(srcor);
end;
procedure ert (f, t, ai : w;
         notj : bl);
 var
  k : w;
begin
 for k := 1 to 32 do
  begin
   if bt(z.kn[ai], k) then
    fa(z.l[-k], z.l[ai], psz, tsz, notj);
   if bt(z.ou[ai], k) and notj then
    fa(z.l[ai], z.l[-k], tsz, psz, notj);
  end;
end;
procedure yl;
begin
 hideall;
 sdr;
 c.h := 30;
 c.v := 50;
 frameoval(prect(c, 1));
 pm(c, '-->', 'n', 'p');
 c.v := 100;
 frameoval(prect(c, 1));
 pm(c, '->->', 'n', 'p');
 c.v := 150;
 frameoval(prect(c, 1));
 pm(c, 'Key', 'n', 'p');
 c.v := 200;
 frameoval(prect(c, 1));
 pm(c, 'EOS', 'n', 'p');
 moveto(60, 20);
 lineto(60, 300);
end;
procedure redraw;
 var
```

```
  i : w;
begin
 yl;
 textface([bold]);
 for i := 1 to z.p do
  begin
   pm(z.l[-i], z.na[-i], 'n', 'p');
   frameoval(prect(z.l[-i], i));
  end;
 for i := 1 to z.t do
  begin
   paintrect(prect(z.l[i], 2));
   ert(0, 0, i, tt);
   pm(z.l[i], z.na[i], 'n', 't');
  end;
 for i := z.mf + 1 to z.mf + z.m do
  begin
   paintrect(prect(z.l[i], 2));
   eraserect(prect(z.l[i], 3));
   ert(0, 0, i, tt);
   pm(z.l[i], z.na[i], 'n', 't');
  end;
 for i := z.mf + z.mm + 1 to z.mf + z.mm + z.j do
  begin
   framerect(prect(z.l[i], 4));
   ert(0, 0, i, ff);
   pm(z.l[i], z.na[i], 'n', 't');
  end;
 textmode(srcor);
end;
procedure M (i : w;
             t1 : w);
 var
  c : w;
begin
 if t1 = 1 then
  begin
   sx;
   for c := i to i + 2 do
    writeln(mt.mg[c]);
  end
```

```
  else if t1 = 0 then
   writeln(y, mt.mg[i]);
end;
procedure pln;
begin
 M(190, 0);
end;
procedure init;
begin
 Flushevents($017F, 0);
 OldMenuBar := Pointer(LInLineF($A93B));
 InLineP($A934);
 fm := Pointer(LInLineF($A931, 100, 'UTILITIES      '));
 InLineP($A933, fm, 'DISPLAY DATABASE;CLEAR;PRINT
      DATABASE;NAME;QUIT;HELP');
 InLineP($A935, fm, 0);
 am := Pointer(LInLineF($A931, 102, 'ANALYSIS    '));
 InLineP($A933, am, 'IMPLICATION;CLOSURE;KEY;HELP ');
 InLineP($A935, am, 0);
 sm := Pointer(LInLineF($A931, 101, 'DATABASE MODIFICATION'));
 InLineP($A933, sm, 'ADD ARC;ADD FD;ADD MVD;ADD SCHEME;ADD
      ATTRIBUTE');
 InLineP($A935, sm, 0);
 um := pointer(linlinef($a931, 103, 'FILE UTILITIES    '));
 inlinep($a933, um, 'CREATE DATABASE;SAVE PICTURE;ACCEPT
      MODIFICATION;DELETE ARC;HELP DB. MOD.;HELP');
 inlineP($a935, um, 0);
 inlinep($a937);
end;

procedure sb (var long : q;
         pos : w;
         bt : bl);
 var
  t : q;
begin
 t := 1;
 t := bitshift(t, pos - 1);
 if not bt then
  begin
   t := bitnot(t);
```

```
    long := bitand(long, t);
   end
  else
   long := bitor(long, t);
end;
function il (i, j : q) : bl;
 var
  c : q;
begin
 il := tt;
 for c := 1 to z.p do
  if bt(j, c) then
   if not bt(i, c) then
    il := ff;
end;
procedure cnct (var i, j : q);
 var
  c : q;
begin
 for c := 1 to z.p do
  if bt(j, c) then
   if not bt(i, c) then
    sb(i, c, tt);
end;
function ir (i, j : q) : q;
 var
  c : w;
  m : q;
begin
 m := 0;
 for c := 1 to z.p do
  if (bt(i, c) = tt) and (bt(j, c) = tt) then
   sb(m, c, tt);
 ir := m;
end;
function cmp (cv : q) : q;
 var
  c : w;
  j : q;
begin
 j := 0;
```

```
 for c := 1 to z.p do
  if not bt(cv, c) then
   sb(j, c, tt);
 cmp := j;
end;
procedure fd (at : str10;
        var pt : w);
 var
  c : w;
begin
 pt := 0;
 for c := 1 to z.p do
  begin
   if z.na[-c] = at then
    pt := c;
  end;
end;
procedure ab (s : str256;
        var cv : q);
 var
  pl, pt : w;
  at : str10;
begin
 pl := 1;
 pt := 1;
 cv := 0;
 pl := pos(',', s);
 if pl = 0 then
  begin
   at := copy(s, 1, 10);
   fd(at, pt);
   sb(cv, pt, tt);
  end;
 while (pl <> 0) do
  begin
   if (pl <> 0) and (s <> ' ') then
    begin
     at := ' ';
     at := copy(s, 1, pl - 1);
     fd(at, pt);
     if pt <> 0 then
```

```
        begin
         sb(cv, pt, tt);
         delete(s, 1, pl);
        end;
      end;
     pl := pos(',', s);
    end;
   at := copy(s, 1, 10);
   fd(at, pt);
   if pt <> 0 then
    sb(cv, pt, tt);
end;
procedure df (p : point);
begin
 if z.t < z.mf then
  begin
   paintrect(prect(p, 2));
   z.t := z.t + 1;
   z.l[z.t] := p;
  end
 else
  note(2500, 20, 10);
end;
procedure dm (p : point);
begin
 if z.m < z.mm then
  begin
   paintrect(prect(c, 2));
   eraserect(prect(c, 3));
   z.m := z.m + 1;
   z.l[z.mf + z.m] := p;
  end
 else
  note(2500, 20, 10);
end;
procedure nwn;
 var
  iz : net;
begin
 z := iz;
 z.mxp := 32;
```

```
 M(151, 1);
 readln(z.mf);
 M(154, 1);
 readln(z.mm);
 M(157, 1);
 readln(z.mj);
 z.id := 1;
 yl;
end;
procedure drp (p : point);
begin
 frameoval(prect(p, 1));
 z.p := z.p + 1;
 z.l[-z.p].h := p.h;
 z.l[-z.p].v := p.v;
end;
procedure dj (p : point);
begin
 if z.j < z.mj then
  begin
   framerect(prect(p, 4));
   z.j := z.j + 1;
   z.l[z.mf + z.mm + z.j] := p;
  end
 else
  note(2500, 20, 10);
end;
procedure arc (blk : bl);
 var
  aii, bjj : w;
begin
 if fir then
  begin
   a := fnt(c.h, c.v, ai);
   if a > 0 then
    begin
     fir := ff;
     note(1000, 10, 5);
    end;
  end
 else
```

```
begin
 getmouse(c.h, c.v);
 b := fnt(c.h, c.v, bj);
 if b > 0 then
  begin
   if not blk then
    penmode(patbic);
   fir := tt;
   case a + b of
    4,-8 :
     if a = 1 then
      begin
       fa(z.l[ai], z.l[bj], psz, tsz, tt);
       sb(z.kn[bj], -ai, blk);
       if bt(z.ou[bj], -ai) then
        begin
         pennormal;
         fa(z.l[bj], z.l[ai], tsz, psz, tt);
        end;
      end
     else
      begin
       fa(z.l[ai], z.l[bj], tsz, psz, tt);
       sb(z.ou[ai], -bj, blk);
       if bt(z.kn[ai], -bj) then
        begin
         pennormal;
         fa(z.l[bj], z.l[ai], psz, tsz, tt);
        end;
      end;
    16 :
     begin
      aii := ai;
      bjj := bj;
      if b = 1 then
       begin
        bjj := ai;
        aii := bj;
        fir := ff;
       end;
      fa(z.l[aii], z.l[bjj], psz, tsz, ff);
```

```
          sb(z.kn[bjj], -aii, blk);
         end;
        otherwise
         ;
       end;
       pennormal;
       note(1000, 10, 8);
       note(1500, 10, 5);
      end;
    end;
end;
procedure npa;
 var
  k, ki : w;
begin
 getmouse(c.h, c.v);
 k := fnt(c.h, c.v, ki);
 if k <> 0 then
  begin
   invertrect(prect(z.l[ki], 1));
   sx;
   M(184, 1);
   readln(z.na[ki]);
   invertrect(prect(z.l[ki], 1));
  end;
end;
function fdi (i : w) : str10;
begin
 fdi := z.na[-i];
end;
procedure ba (var att : str256;
         cv : q;
         i : w);
 var
  c : w;
begin
 att := ' ';
 for c := 1 to z.p do
  if bt(cv, c) then
   att := concat(att, fdi(c), ' ');
 case i of
```

```
    0 :
     writeln(att);
    1 :
     writeln(y, att);
    3 :
     write(y, att);
   end;
  end;
  procedure cln;
   var
    i : w;
  begin
   for i := 1 to x do
    dt[i] := 0;
  end;

  procedure uo (i, j : q;
          var k : q);
   var
    c : w;
  begin
   k := 0;
   for c := 1 to z.p do
    if bt(j, c) or bt(i, c) then
     sb(k, c, tt);
  end;
  function nk (z : w) : bl;
  begin
   nk := tt;
   if (sqr(c.h - 30) + sqr(c.v - 50 * z) <= psz * psz) then
    nk := ff;
  end;
  procedure ga (var s : q);
   var
    k, ki : w;
  begin
   getmouse(c.h, c.v);
   k := fnt(c.h, c.v, ki);
   if k <> 0 then
    sb(s, -ki, tt);
  end;
```

```
procedure pdb;
 var
  i, ct, st : w;
  sr, zst : str10;
  a : str256;
begin
 M(160, 0);
 for i := 1 to z.p do
  write(y, z.na[-i], '  ');
 wy;
 pln;
 M(187, 0);
 if z.t <> 0 then
  for i := 1 to z.t do
    begin
     write(y, z.na[i], '  :  ');
     ba(a, z.kn[i], 3);
     write(y, '-->');
     ba(a, z.ou[i], 1);
     wy;
    end;
 if z.m <> 0 then
  for i := z.mf + 1 to z.m + z.mf do
    begin
     write(y, z.na[i], '  :  ');
     ba(a, z.kn[i], 3);
     write(y, '->->');
     ba(a, z.ou[i], 1);
     wy;
    end;
 if z.j <> 0 then
  begin
    ct := 0;
    st := z.mf + z.mm + 1;
    sr := z.na[st];
    while (ct <= z.j) and (st <= z.mf + z.mm + z.j) do
     begin
       zst := z.na[st];
       write(y, zst, '  :  ');
       write(y, '  (   ');
       while (sr = zst) do
```

```
      begin
       write(y, '   ,   ');
       ba(a, z.kn[st], 3);
       st := st + 1;
       ct := ct + 1;
       if (st <= z.mf + z.mm + z.mj) then
        sr := z.na[st];
      end;
     sr := z.na[st];
     zst := z.na[st];
     writeln(y, '   )   ');
    end;
  end;
 pln;
end;
procedure ms (a, b : q;
        var c : q);
 var
  i : w;
begin
 c := 0;
 for i := 1 to z.p do
  if (bt(a, i)) and (not (bt(b, i))) then
   sb(c, i, tt);
end;
function un (tp : q) : bl;
 var
  tp1 : q;
  i : w;
begin
 un := ff;
 for i := 1 to x do
  if il(tp, dt[i]) then
   begin
    ms(tp, dt[i], tp1);
    tp := tp1;
   end;
 if tp = 0 then
  un := tt;
end;
procedure db (var x : w;
```

```
        cv : q);
 var
  fl : bl;
  xx : str256;
  i, j : w;
  ln : q;
begin
 x := 1;
 for i := 1 to z.p do
  begin
   ln := 0;
   if bt(cv, i) then
    begin
     sb(ln, i, tt);
     dt[x] := ln;
     x := x + 1;
    end;
  end;
 dt[x] := cmp(cv);
 fl := tt;
 while fl do
  begin
   fl := ff;
   for i := 1 to mvd do
    for j := 1 to x do
     if (not (il(op[i], dt[j]))) and (ir(dt[j], op[i]) <> 0) and
          (ir(dt[j], ip[i]) = 0) then
       begin
        x := x + 1;
        ms(dt[j], op[i], dt[x]);
        dt[j] := ir(dt[j], op[i]);
        fl := tt;
       end;
  end;
end;
procedure tr;
 var
  v, tp : q;
  i, c, d, ps, ct, nm, st : w;
  sr, zst : str10;
begin
```

```
i := 0;
v := 0;
if z.t <> 0 then
 for c := 1 to z.t do
  for d := 1 to z.p do
   if bt(z.ou[c], d) then
    begin
     i := i + 1;
     sb(v, d, tt);
     ip[i] := z.kn[c];
     op[i] := v;
     v := 0;
    end;
if z.m <> 0 then
 for c := 1 + z.mf to z.m + z.mf do
  begin
   i := i + 1;
   ip[i] := z.kn[c];
   op[i] := z.ou[c];
  end;
if z.j <> 0 then
 begin
  ct := 0;
  st := z.mf + z.mm + 1;
  ps := st;
  sr := z.na[st];
  nm := 0;
  while (ct <= z.j) and (st <= z.mf + z.mm + z.j) do
   begin
    zst := z.na[st];
    while sr = zst do
     begin
      st := st + 1;
      ct := ct + 1;
      nm := nm + 1;
      if (st <= z.mf + z.mm + z.mj) then
       sr := z.na[st];
     end;
    if nm = 2 then
     begin
      i := i + 1;
```

```
        ip[i] := ir(z.kn[ps], z.kn[ps + 1]);
        ms(z.kn[ps], ip[i], op[i]);
        if (ip[i] = 0) or (op[i] = 0) then
         i := i - 1;
        ps := ps + 2;
        nm := 0;
       end
      else if nm = 3 then
       begin
        uo(z.kn[ps + 1], z.kn[ps + 2], tp);
        i := i + 1;
        ip[i] := ir(z.kn[ps], tp);
        ms(z.kn[ps], ip[i], op[i]);
        if (ip[i] = 0) or (op[i] = 0) then
         i := i - 1;
        uo(z.kn[ps], z.kn[ps + 2], tp);
        i := i + 1;
        ip[i] := ir(z.kn[ps + 1], tp);
        ms(z.kn[ps + 1], ip[i], op[i]);
        if (ip[i] = 0) or (op[i] = 0) then
         i := i - 1;
        uo(z.kn[ps + 1], z.kn[ps], tp);
        i := i + 1;
        ip[i] := ir(z.kn[ps + 2], tp);
        ms(z.kn[ps + 2], ip[i], op[i]);
        if (ip[i] = 0) or (op[i] = 0) then
         i := i - 1;
        ps := ps + 3;
        nm := 0;
       end;
     end;
   end;
  mvd := i;
end;
function cl (a : q;
        i : w) : q;
 var
  fl : bl;
  lcl, cls, k, cvc : q;
  j, c : w;
begin
```

```
cls := a;
lcl := 0;
if i = 0 then
 while cls <> lcl do
  begin
   lcl := cls;
   for c := 1 to z.t do
    if il(cls, z.kn[c]) then
     cnct(cls, z.ou[c]);
  end
 else
 begin
  k := 0;
  fl := ff;
  cvc := cmp(cls);
  for c := 1 to z.p do
   begin
    if bt(cvc, c) then
     begin
      sb(k, c, tt);
      for j := 1 to x do
       if k = dt[j] then
        fl := tt;
      if fl = tt then
       for j := 1 to z.t do
        if z.ou[j] = k then
         cnct(cls, k);
     end;
    fl := ff;
    k := 0;
   end;
  end;
 cl := cls;
end;
procedure impl (t : w;
         cv, rg : q);
 var
  lf, ty, xx : str256;
  pl, i : w;
  ex : bl;
  tp, cls : q;
```

```
begin
 if t = 0 then
  begin
   if (z.m = 0) and (z.j = 0) then
    begin
     cls := cl(cv, 0);
     if il(cls, rg) then
      ex := tt;
    end
   else
    begin
     db(x, cv);
     if un(rg) then
      for i := 1 to z.t do
       begin
        ms(z.ou[i], z.kn[i], tp);
        if il(tp, rg) then
         ex := tt;
       end;
    end;
   if ex = tt then
    M(163, 0)
   else
    M(166, 0);
  end
 else if t = 1 then
  begin
   db(x, cv);
   if un(rg) then
    M(163, 0)
   else
    M(166, 0);
  end;
 cln;
end;
function Key (a, b : q) : bl;
 var
  k : bl;
  cls : q;
begin
 k := ff;
```

```
 cls := 0;
 if (z.m = 0) and (z.j = 0) then
  cls := cl(a, 0)
 else
  begin
   db(x, a);
   cls := cl(a, 1);
  end;
 if il(cls, b) and il(b, a) then
  k := tt;
 key := k;
end;
procedure dn;
 var
  i, j : w;
  a : str256;
begin
 ba(att, clos, 1);
 pln;
 M(88, 0);
 M(175, 0);
 for i := 1 to x do
  for j := 1 to z.p do
   if bt(dt[i], j) then
    z.tn[-j, 1] := i;
 for i := 1 to x do
  begin
   write(y, '>  ');
   ba(a, dt[i], 1);
  end;
 pln;
 cln;
 redraw;
 nt := tt;
 dsptk;
end;
procedure dc;
begin
 write(y, 'Testing whether : ');
 ba(att, cv, 3);
 write(y, '  is a key of  :');
```

```
 ba(att, cv1, 1);
 if key(cv, cv1) then
  m(169, 0)
 else
  m(172, 0);
 cln;
 pln;
end;
procedure fkey (a : q;
         var b : q);
 var
  fl : bl;
  i : w;
  c : q;
begin
 fl := ff;
 b := 0;
 for i := 1 to z.t do
  if key(z.kn[i], a) then
   begin
    fl := tt;
    b := z.kn[i];
    ba(att, b, 1);
   end;
 if not fl then
  while (not fl) and (i <= z.t) do
   begin
    i := i + 1;
    uo(b, z.kn[i], c);
    b := c;
    if key(a, b) then
     begin
      ba(att, b, 1);
      fl := tt;
     end;
   end;
end;
procedure handlemenu;
begin
 case WInLineF($A86A, nm) of
  100 :
```

```
case WInLineF($A86B, nm) of
 1 :
  redraw;
 2 :
  hideall;
 3 :
  pdb;
 4 :
  npa;
 5 :
  begin
   M(79, 1);
   readln(att);
   sv := ff;
   if att = 'y' then
    sv := tt;
   M(82, 1);
   readln(att);
   done := tt;
   if att = 'y' then
    nomore := tt;
  end;
 6 :
  ;
end;
102 :
 case WInLineF($A86B, nm) of
  1 :
   begin
    ga(cv);
    if nk(1) = ff then
     begin
      t1 := 0;
      cv1 := cv;
      cv := 0;
     end
    else if nk(2) = ff then
     begin
      t1 := 1;
      cv1 := cv;
      cv := 0;
```

```
    end;
   if nk(4) = ff then
    begin
     m(91, 0);
     ba(att, cv1, 3);
     if t1 = 1 then
      write(y, '  ->-> ')
     else
      write(y, ' -->  ');
     ba(att, cv, 1);
     m(175, 0);
     impl(t1, cv1, cv);
     pln;
    end;
  end;
 2 :
  begin
   ga(cv);
   if nk(4) = ff then
    begin
     m(85, 0);
     ba(att, cv, 1);
     m(175, 0);
     db(x, cv);
     if (z.m = 0) and (z.j = 0) then
      clos := cl(cv, 0)
     else
      clos := cl(cv, 1);
     dn;
    end;
  end;
 3 :
  begin
   ga(cv);
   if nk(3) = ff then
    begin
     cv1 := cv;
     cv := 0;
    end;
   if nk(4) = ff then
    if cv <> 0 then
```

```
        dc
       else
        begin
         m(67, 0);
         ba(att, cv1, 1);
         m(175, 0);
         fkey(cv1, cv);
         cln;
         pln;
        end;
      end;
     4 :
      begin
       m(13, 1);
       readln(n);
       case n of
        1 :
         m(22, 1);
        2 :
         m(16, 1);
        3 :
         m(10, 1);
        4 :
         m(25, 1);
        otherwise
         m(103, 1);
       end;
      end;
    end;
   101 :
    if not ib then
     begin
      getmouse(c.h, c.v);
      case WInLineF($A86B, nm) of
       1 :
        ARC(TT);
       2 :
        df(c);
       3 :
        DM(c);
       4 :
```

```
      DJ(c);
     5 :
      drp(c);
    end;
   end;
103 :
 case WInLineF($A86B, nm) of
  1 :
   begin
    dbfile := newfilename(pmp);
    nwn;
   end;
  2 :
   begin
    M(46, 1);
    readln(pc);
    hideall;
    redraw;
    dsptk;
    savedrawing(pc);
   end;
  3 :
   begin
    M(97, 1);
    readln(att);
    if att = 'y' then
     begin
      open(dbvar, dbfile);
      dbvar^ := z;
      put(dbvar);
      close(dbvar);
     end;
    tr;
   end;
  4 :
   begin
    getmouse(c.h, c.v);
    arc(ff);
   end;
  5 :
   ;
```

```
      6 :
       ;
     end;
    otherwise
     ;
  end;
 end;(*proc*)
begin
 rewrite(y, 'printer:');
 write(y, chr(27), 'c', chr(27), 'p');
 write(y, chr(29), 'A@');
 for delay := 1 to 58 do
  write(y, '@@');
 write(y, 'C@@@@@@@@@@@@@@A@', chr(30));
 psz := 10;
 tsz := 8;
 nomore := ff;
 nt := ff;
 fir := ff;
 om := 999999;
 scr('WELCOME TO  G L O R D');
 for delay := 1 to 3000 do
  ;
 init;
 hideall;
 while nomore = ff do
  begin
   done := ff;
   open(msgvar, 'msgsI');
   mt := msgvar^;
   close(msgvar);
   pmp := ' database file :';
   dbfile := oldfilename(pmp);
   if dbfile = '' then
    begin
     dbfile := newfilename(pmp);
     nwn;
    end
   else
    begin
     open(dbvar, dbfile);
```

```
      z := dbvar^;
      close(dbvar);
      tr;
      pln;
      M(178, 0);
      pln;
      writeln(y, 'Given : the database is:   ', dbfile);
      pln;
      writeln(y, '  A N A L Y S I S      P R O B L E M S ');
      pln;
     end;
    repeat
     if testmouse(ib) then
      begin
       if ib then
        begin
         fir := tt;
         cv := 0;
         cv1 := 0;
         om := LinLineF($A93D, ev.wr);
        end;
       nm := om;
       Handlemenu;
      end;
    until done;
    M(181, 0);
    pln;
    if sv then
     begin
      open(dbvar, dbfile);
      dbvar^ := z;
      put(dbvar);
      close(dbvar);
     end;
   end;
  close(y);
  InlineP($A932, sm);
  InLineP($A93C, OldMenuBar);
  InLineP($A937);
  scr('BYE');
end.
```

```
program GLORD2;
 uses
  quickdraw1, quickdraw2;
 const
  TT = true;
  FF = false;
  mx = 32;
 type
  str10 = string[10];
  str256 = string[80];
  w = INTEGER;
  bl = boolean;
  q = longint;
  Ptr = ^q;
  Handle = ^Ptr;
  WindowRecord = array[1..78] of w;
  WindowPtr = ^WindowRecord;
  str24 = string[24];
  str32 = string[32];
  evr = record
    wt : w;
    message : q;
    wn : q;
    wr : q;
    modifiers : w;
   end;
  mgs = record
    mg : array[1..250] of string[80];
   end;
  net = record
    id : w;
    p, t, m, j, c, mxp, mf, mm, mj, mxc : 0..64;
    na : array[-32..mx] of string[16];
    para, act : array[-32..mx] of str32;
    l : array[-32..mx] of point;
    token : array[-32..mx, 1..4] of w;
    kn, ou : array[1..mx] of q;
   end;
 var
  done, ib, fir, rec, nomore, sv, nt : bl;
```

```
 ev : evr;
 pc : str24;
 r : rect;
 att, at1 : str256;
 delay, n, rc, x, mvd, tsz, psz, a, b, ai, bj, t1, dy : w;
 om, nm, cv, clos, cv1, uni, u1 : q;
 op, ip, dt : array[1..30] of q;
 gm : array[1..30, 1..10] of w;
 fd : array[1..20} of w;
 whichwindow : windowptr;
 z : net;
 c : point;
 pmp, dbfile : str24;
 dbvar : file of net;
 msgvar : file of mgs;
 mt : mgs;
 y : text;
 OldMenuBar, fm, am, sm, um : Handle;
procedure wy;
begin
 writeln(y);
end;
procedure sdr;
begin
 hideall;
 setrect(R, 0, 30, 527, 357);
 setdrawingrect(R);
 showdrawing;
end;
function testmouse (var bar : bl) : bl;
begin
 testmouse := ff;
 bar := ff;
 repeat
 until BinLineF($A970, $017F, @ev);
 if ev.wt=>1 then
  begin
   testmouse := tt;
   if WinLineF($A92C, ev.wr, @whichwindow) = 1 then
    bar := tt;
  end;
```

```
end;
procedure sx;
begin
 setrect(r, 0, 30, 521, 100);
 settextrect(r);
 showtext;
end;
procedure scr (m : str32);
begin
 sdr;
 fillrect(0, 0, 322, 512, Gray);
 setrect(r, 40, 150, 472, 230);
 fillroundrect(r, 20, 20, white);
 frameroundrect(r, 20, 20);
 textface([bold, outline]);
 moveto(150, 200);
 drawstring(m);
end;
function bt (bts, n : q) : bl;
begin
 if odd(bitshift(bts, 1 - n)) then
  bt := tt
 else
  bt := ff;
end;
procedure M (i : w;
             t1 : w);
 var
  c : w;
begin
 if t1 = 1 then
  begin
   sx;
   for c := i to j + 2 do
    writeln(mt.mg[c]);
  end
 else if t1 = 0 then
  writeln(y, mt.mg[i]);
end;
procedure pln;
begin
```

```
 ev : evr;
 pc : str24;
 r : rect;
 att, at1 : str256;
 delay, n, rc, x, mvd, tsz, psz, a, b, ai, bj, t1, dy : w;
 om, nm, cv, clos, cv1, uni, u1 : q;
 op, ip, dt : array[1..30] of q;
 gm : array[1..30, 1..10] of w;
 fd : array[1..20] of w;
 whichwindow : windowptr;
 z : net;
 c : point;
 pmp, dbfile : str24;
 dbvar : file of net;
 msgvar : file of mgs;
 mt : mgs;
 y : text;
 OldMenuBar, fm, am, sm, um : Handle;
procedure wy;
begin
 writeln(y);
end;
procedure sdr;
begin
 hideall;
 setrect(R, 0, 30, 527, 357);
 setdrawingrect(R);
 showdrawing;
end;
function testmouse (var bar : b1) : b1;
begin
 testmouse := ff;
 bar := ff;
 repeat
 until BinLineF($A970, $017F, @ev);
 if ev.wt = 1 then
  begin
   testmouse := tt;
   if WinLineF($A92C, ev.wr, @whichwindow) = 1 then
    bar := tt;
  end;
```

```
end;
procedure sx;
begin
 setrect(r, 0, 30, 521, 100);
 settextrect(r);
 showtext;
end;
procedure scr (m : str32);
begin
 sdr;
 fillrect(0, 0, 322, 512, Gray);
 setrect(r, 40, 150, 472, 230);
 fillroundrect(r, 20, 20, white);
 frameroundrect(r, 20, 20);
 textface([bold, outline]);
 moveto(150, 200);
 drawstring(m);
end;
function bt (bts, n : q) : bl;
begin
 if odd(bitshift(bts, 1 - n)) then
  bt := tt
 else
  bt := ff;
end;
procedure M (i : w;
         t1 : w);
 var
  c : w;
begin
 if t1 = 1 then
  begin
   sx;
   for c := i to i + 2 do
    writeln(mt.mg[c]);
  end
 else if t1 = 0 then
  writeln(y, mt.mg[i]);
end;
procedure pln;
begin
```

```
 M(190, 0);
end;
procedure init;
begin
 Flushevents($017F, 0);
 OldMenuBar := Pointer(LInLineF($A93B));
 InLineP($A934);
 fm := Pointer(LInLineF($A931, 100, 'UTILITIES      '));
 InLineP($A933, fm, 'CLEAR;PRINT DATABASE;QUIT;HELP');
 InLineP($A935, fm, 0);
 am := Pointer(LInLineF($A931, 102, 'ANALYSIS     '));
 InLineP($A933, am, 'IMPLICATION;CLOSURE;KEY;HELP ');
 InLineP($A935, am, 0);
 sm := Pointer(LInLineF($A931, 101, 'NORMAL FORMS    '));
 InLineP($A933, sm, 'SYNTHESIZE 3NF;TEST 3NF;GENERATE BCNF;TEST
      4NF;GENERATE 4NF;TEST ACYCLICITY;HELP');
 InLineP($A935, sm, 0);
 um := pointer(linlinef($a931, 103, 'FILE UTILITIES  '));
 inlinep($a933, um, 'RECORD ON;RECORD OFF;PRINT FILE;HELP');
 inlineP($a935, um, 0);
 inlinep($a937);
end;

procedure sb (var long : q;
         pos : w;
         bt : bl);
 var
  t : q;
begin
 t := 1;
 t := bitshift(t, pos - 1);
 if not bt then
  begin
   t := bitnot(t);
   long := bitand(long, t);
  end
 else
  long := bitor(long, t);
end;
function il (i, j : q) : bl;
 var
```

```
  c : q;
begin
 il := tt;
 for c := 1 to z.p do
  if bt(j, c) then
   if not bt(i, c) then
     il := ff;
end;
procedure cnct (var i, j : q);
 var
  c : q;
begin
 for c := 1 to z.p do
  if bt(j, c) then
   if not bt(i, c) then
     sb(i, c, tt);
end;
function ir (i, j : q) : q;
 var
  c : w;
  m : q;
begin
 m := 0;
 for c := 1 to z.p do
  if (bt(i, c) = tt) and (bt(j, c) = tt) then
   sb(m, c, tt);
 ir := m;
end;
function cmp (cv : q) : q;
 var
  c : w;
  j : q;
begin
 j := 0;
 for c := 1 to z.p do
  if not bt(cv, c) then
   sb(j, c, tt);
 cmp := j;
end;
procedure find (at : str10;
          var pt : w);
```

```
 var
  c : w;
begin
 pt := 0;
 for c := 1 to z.p do
  begin
   if z.na[-c] = at then
    pt := c;
  end;
end;
procedure ab (s : str256;
        var cv : q);
 var
  pl, pt : w;
  at : str10;
begin
 pl := 1;
 pt := 1;
 cv := 0;
 pl := pos(',', s);
 if pl = 0 then
  begin
   at := copy(s, 1, 10);
   find(at, pt);
   sb(cv, pt, tt);
  end;
 while (pl <> 0) do
  begin
   if (pl <> 0) and (s <> ' ') then
    begin
     at := ' ';
     at := copy(s, 1, pl - 1);
     find(at, pt);
     if pt <> 0 then
      begin
       sb(cv, pt, tt);
       delete(s, 1, pl);
      end;
    end;
   pl := pos(',', s);
  end;
```

```
 at := copy(s, 1, 10);
 find(at, pt);
 if pt <> 0 then
  sb(cv, pt, tt);
end;
function findi (i : w) : str10;
begin
 findi := z.na[-i];
end;
procedure ba (var att : str256;
        cv : q;
        i : w);
 var
  c : w;
begin
 att := ' ';
 for c := 1 to z.p do
  if bt(cv, c) then
   att := concat(att, findi(c), ' ');
 case i of
  0 :
   writeln(att);
  1 :
   writeln(y, att);
  3 :
   write(y, att);
 end;
end;
procedure cln;
 var
  i : w;
begin
 for i := 1 to x do
  dt[i] := 0;
end;

procedure uo (i, j : q;
        var k : q);
 var
  c : w;
begin
```

```
 k := 0;
 for c := 1 to z.p do
  if bt(j, c) or bt(i, c) then
   sb(k, c, tt);
end;
procedure pdb;
 var
  i, ct, st : w;
  sr, zst : str10;
  a : str256;
begin
 M(160, 0);
 for i := 1 to z.p do
  write(y, z.na[i], '  ');
 wy;
 pln;
 M(187, 0);
 if z.t <> 0 then
  for i := 1 to z.t do
   begin
    write(y, z.na[i], '  :  ');
    ba(a, z.kn[i], 3);
    write(y, '-->');
    ba(a, z.ou[i], 1);
    wy;
   end;
 if z.m <> 0 then
  for i := z.mf + 1 to z.m + z.mf do
   begin
    write(y, z.na[i], '  :  ');
    ba(a, z.kn[i], 3);
    write(y, '->->');
    ba(a, z.ou[i], 1);
    wy;
   end;
 if z.j <> 0 then
  begin
   ct := 0;
   st := z.mf + z.mm + 1;
   sr := z.na[st];
   while (ct <= z.j) and (st <= z.mf + z.mm + z.j) do
```

```
    begin
     zst := z.na[st];
     write(y, zst, '  :  ');
     write(y, '                    (   ');
     while (sr = zst) do
      begin
       write(y, '                        ,  ');
       ba(a, z.kn[st], 1);
       st := st + 1;
       ct := ct + 1;
       if (st <= z.mf + z.mm + z.mj) then
        sr := z.na[st];
      end;
     sr := z.na[st];
     zst := z.na[st];
     write(y, '                          )  ');
     wy;
    end;
  end;
 pln;
end;
procedure ms (a, b : q;
          var c : q);
 var
  i : w;
begin
 c := 0;
 for i := 1 to z.p do
  if (bt(a, i)) and (not (bt(b, i))) then
   sb(c, i, tt);
end;
function un (tp : q) : bl;
 var
  tp1 : q;
  i : w;
begin
 un := ff;
 for i := 1 to x do
  if il(tp, dt[i]) then
   begin
    ms(tp, dt[i], tp1);
```

```
    tp := tp1;
   end;
 if tp = 0 then
  un := tt;
end;
procedure db (var x : w;
        cv : q);
 var
  fl : bl;
  xx : str256;
  i, j : w;
  in : q;
begin
 x := 1;
 for i := 1 to z.p do
  begin
   ln := 0;
   if bt(cv, i) then
   begin
     sb(ln, i, tt);
     dt[x] := ln;
     x := x + 1;
    end;
  end;
 dt[x] := cmp(cv);
 fl := tt;
 while fl do
  begin
   fl := ff;
   for i := 1 to mvd do
    for j := 1 to x do
     if (not (il(op[i], dt[j]))) and (ir(dt[j], op[i]) <> 0) and
          (ir(dt[j], ip[i]) = 0) then
       begin
        x := x + 1;
        ms(dt[j], op[i], dt[x]);
        dt[j] := ir(dt[j], op[i]);
        fl := tt;
       end;
  end;
end;
```

```
procedure tr;
 var
  v, tp : q;
  i, c, d, ps, ct, nm, st : w;
  sr, zst : str10;
begin
 i := 0;
 v := 0;
 if z.t <> 0 then
  for c := 1 to z.t do
   for d := 1 to z.p do
    if bt(z.ou[c], d) then
     begin
      i := i + 1;
      sb(v, d, tt);
      ip[i] := z.kn[c];
      op[i] := v;
      v := 0;
     end;
 if z.m <> 0 then
  for c := 1 + z.mf to z.m + z.mf do
   begin
    i := i + 1;
    ip[i] := z.kn[c];
    op[i] := z.ou[c];
   end;
 if z.j <> 0 then
  begin
   ct := 0;
   st := z.mf + z.mm + 1;
   ps := st;
   sr := z.na[st];
   nm := 0;
   while (ct <= z.j) and (st <= z.mf + z.mm + z.j) do
    begin
     zst := z.na[st];
     while sr = zst do
      begin
       st := st + 1;
       ct := ct + 1;
       nm := nm + 1;
```

```
      if (st <= z.mf + z.mm + z.mj) then
       sr := z.na[st];
     end;
    if nm = 2 then
     begin
      i := i + 1;
      ip[i] := ir(z.kn[ps], z.kn[ps + 1]);
      ms(z.kn[ps], ip[i], op[i]);
      if (ip[i] = 0) or (op[i] = 0) then
       i := i - 1;
      ps := ps + 2;
      nm := 0;
     end
    else if nm = 3 then
     begin
      uo(z.kn[ps + 1], z.kn[ps + 2], tp);
      i := i + 1;
      ip[i] := ir(z.kn[ps], tp);
      ms(z.kn[ps], ip[i], op[i]);
      if (ip[i] = 0) or (op[i] = 0) then
       i := i - 1;
      uo(z.kn[ps], z.kn[ps + 2], tp);
      i := i + 1;
      ip[i] := ir(z.kn[ps + 1], tp);
      ms(z.kn[ps + 1], ip[i], op[i]);
      if (ip[i] = 0) or (op[i] = 0) then
       i := i - 1;
      uo(z.kn[ps + 1], z.kn[ps], tp);
      i := i + 1;
      ip[i] := ir(z.kn[ps + 2], tp);
      ms(z.kn[ps + 2], ip[i], op[i]);
      if (ip[i] = 0) or (op[i] = 0) then
       i := i - 1;
      ps := ps + 3;
      nm := 0;
     end;
   end;
  end;
 mvd := i;
end;
function cl (a : q;
```

```
        i : w) : q;
var
 fl : bl;
 lcl, cls, k, cvc : q;
 j, c : w;
begin
 cls := a;
 lcl := 0;
 if i = 0 then
  while cls <> lcl do
   begin
    lcl := cls;
    for c := 1 to z.t do
     if il(cls, z.kn[c]) then
      cnct(cls, z.ou[c]);
   end
  else
   begin
    k := 0;
    fl := ff;
    cvc := cmp(cls);
    for c := 1 to z.p do
     begin
      if bt(cvc, c) then
       begin
        sb(k, c, tt);
        for j := 1 to x do
         if k = dt[j] then
          fl := tt;
        if fl = tt then
         for j := 1 to z.t do
          if z.ou[j] = k then
           cnct(cls, k);
       end;
      fl := ff;
      k := 0;
     end;
   end;
  cl := cls;
end;
procedure impl;
```

```
var
 lf, ty, xx : str256;
 pl, t, i, e : w;
 ex : bl;
 cv, tp, rg, cls : q;
begin
 pl := pos('-', att);
 ex := ff;
 e := 0;
 if pl = 0 then
  begin
   i := 1;
   M(196, 1);
  end;
 if i = 0 then
  begin
   lf := copy(att, 1, pl - 2);
   ab(lf, cv);
   delete(att, 1, pl - 1);
   ty := copy(att, 1, 3);
   if ty = '-->' then
    begin
     t := 0;
     delete(att, 1, 4);
    end
   else if ty = '->-' then
    begin
     t := 1;
     delete(att, 1, 5);
    end
   else
    begin
     i := 1;
     M(196, 1);
    end;
   if i = 0 then
    begin
     ab(att, rg);
     if t = 0 then
      begin
       if (z.m = 0) and (z.j = 0) then
```

```
        begin
         cls := cl(cv, 0);
         if il(cls, rg) then
          ex := tt;
        end
       else
        begin
         db(x, cv);
         if un(rg) then
          for i := 1 to z.t do
           begin
            ms(z.ou[i], z.kn[i], tp);
            if il(tp, rg) then
             ex := tt;
           end;
        end;
       if ex = tt then
        M(163, 0)
       else
        M(166, 0);
      end
     else if t = 1 then
      begin
       db(x, cv);
       if un(rg) then
        M(163, 0)
       else
        M(166, 0);
      end;
     cln;
    end;
  end;
end;
function Key (a, b : q) : bl;
 var
  k : bl;
  cls : q;
begin
 k := ff;
 cls := 0;
 if (z.m = 0) and (z.j = 0) then
```

```
        begin
         cls := cl(cv, 0);
         if il(cls, rg) then
          ex := tt;
        end
       else
        begin
         db(x, cv);
         if un(rg) then
          for i := 1 to z.t do
           begin
            ms(z.ou[i], z.kn[i], tp);
            if il(tp, rg) then
             ex := tt;
           end;
        end;
       if ex = tt then
        M(163, 0)
       else
        M(166, 0);
      end
     else if t = 1 then
      begin
       db(x, cv);
       if un(rg) then
        M(163, 0)
       else
        M(166, 0);
      end;
     cln;
    end;
  end;
end;
function Key (a, b : q) : bl;
 var
  k : bl;
  cls : q;
begin
 k := ff;
 cls := 0;
 if (z.m = 0) and (z.j = 0) then
```

```
 cls := cl(a, 0)
else
 begin
  db(x, a);
  cls := cl(a, 1);
 end;
if il(cls, b) and il(b, a) then
 k := tt;
key := k;
end;
procedure fkey (a : q;
        var b : q);
var
 fl : bl;
 i : w;
 c, a1, u2 : q;
begin
 fl := ff;
 b := 0;
 ms(a, u1, u2);
 ms(a, u2, a1);
 ba(att, a1, 1);
 ba(att, u1, 1);
 ba(att, u2, 1);
 for i := 1 to z.t do
  if key(z.kn[i], a1) then
   begin
    fl := tt;
    b := z.kn[i];
    uo(b, u2, c);
    ba(att, c, 1);
   end;
 i := 0;
 if not fl then
  while (not fl) and (i < z.t) do
   begin
    i := i + 1;
    uo(b, z.kn[i], c);
    b := c;
    if key(b, a1) then
     begin
```

```
      uo(u2, b, c);
      ba(att, c, 1);
      f1 := tt;
     end;
   end;
 if not f1 then
  writeln(y, 'The key is the relation scheme itself.');
end;
function triv (a, b, c : q) : bl;
 var
  t1, t2, t3 : bl;
  d : q;
  i : w;
begin
 t1 := ff;
 t2 := tt;
 t3 := tt;
 uo(a, b, d);
 for i := 1 to z.p do
  if not (bt(d, i) = bt(c, i)) then
   t2 := ff;
 if not (il(a, b)) then
  t3 := ff;
 if t2 or t3 then
  t1 := tt;
 triv := t1;
end;
function tst4nf (a : q) : bl;
 var
  i : w;
  t1 : bl;
begin
 t1 := tt;
 i := 1;
 while (i <= mvd) and (t1 = tt) do
  begin
   if il(a, ip[i]) and il(a, op[i]) then
    if not triv(ip[i], op[i], a) and not key(ip[i], a) then
     t1 := ff;
   i := i + 1;
  end;
```

```
 tst4nf := tl;
end;
procedure dvd (sc : q;
         var rsc : q);
 var
  i : w;
  stp : bl;
  d, c : q;
begin
 stp := ff;
 i := 0;
 while (not stp) and (i <= z.t) do
  begin
   i := i + 1;
   uo(z.kn[i], z.ou[i], c);
   if il(sc, c) then
    if (not (key(z.kn[i], sc))) and (not (il(z.kn[i], z.ou[i])))
         and (fd[i] <> 99) then
     begin
      ba(att, c, 1);
      writeln(y, 'Key  :    ');
      ba(att, z.kn[i], 1);
      fd[i] := 99;
      stp := tt;
     end;
  end;
 if not stp then
  rsc := sc
 else
  begin
   ms(sc, z.ou[i], d);
   rsc := d;
  end;
end;
procedure gen3nf;
 var
  i, j : w;
  fn, ok : bl;
  sc, cp, scl, cls, c, p, d : q;
begin
 pln;
```

```
m(229, 0);
pln;
p := 0;
fn := ff;
for i := 1 to z.t do
 begin
  uo(z.kn[i], z.ou[i], c);
  uo(c, p, d);
  p := d;
 end;
cp := cmp(p);
if cp <> 0 then
 ms(sc, cp, sc1);
for i := 1 to z.t do
 begin
  uo(z.kn[i], z.ou[i], c);
  if c = sc1 then
   begin
    ok := tt;
    ba(att, sc1, 0);
    ba(att, sc1, 1);
   end;
 end;
if not ok then
 for i := 1 to z.t do
  begin
   uo(z.kn[i], z.ou[i], c);
   fn := ff;
   for j := 1 to i do
    if dt[j] = c then
     fn := tt;
   if fn then
    dt[i] := 0
   else
    dt[i] := c;
  end;
for i := 1 to z.t do
 if dt[i] <> 0 then
  ba(att, dt[i], 1);
cln;
pln;
```

```
end;
procedure genbcnf;
 var
  ok : bl;
  sc, rsc : q;
  i : w;
begin
 for i := 1 to 20 do
  fd[i] := 0;
 pln;
 rsc := 0;
 m(223, 0);
 pln;
 sc := uni;
 ok := ff;
 while not ok do
  begin
   dvd(sc, rsc);
   if sc = rsc then
    ok :=.tt;
   sc := rsc;
  end;
 pln;
end;
function icl (i, k : w) : bl;
 var
  c : w;
begin
 icl := tt;
 for c := 1 to z.p do
  if (gm[c, i] = 1) and (gm[c, k] = 0) then
   icl := ff;
end;
function zc (i : w) : bl;
 var
  c : w;
begin
 zc := tt;
 for c := 1 to z.p do
  if gm[c, i] <> 0 then
   zc := ff;
```

```
end;
procedure gr2 (j : w);
 var
  i, ct, k, p, ps : w;
  fl, fl1, fl2, zm : bl;
begin
 j := j - 1;
 fl := ff;
 fl1 := ff;
 fl2 := ff;
 zm := ff;
 while not fl and not zm do
  begin
   for i := 1 to j do
    for k := 1 to j do
     if i <> k then
      if icl(i, k) and not zc(i) then
       begin
        fl1 := tt;
        for p := 1 to z.p do
         gm[p, i] := 0;
       end;
   for i := 1 to z.p do
    begin
     ct := 0;
     for k := 1 to j do
      if gm[i, k] = 1 then
       begin
        ct := ct + 1;
        ps := k;
       end;
      if ct = 1 then
       begin
        fl2 := tt;
        gm[i, ps] := 0;
       end;
    end;
   fl := tt;
   if fl1 or fl2 then
    fl := ff
   else
```

```
    fl := tt;
   fl1 := ff;
   fl2 := ff;
   zm := tt;
   for i := 1 to z.p do
    for k := 1 to j do
     if gm[i, k] <> 0 then
      zm := ff;
  end;
 zm := tt;
 for i := 1 to z.p do
  for k := 1 to j do
   if gm[i, k] <> 0 then
    zm := ff;
 if zm then
  m(247, 0)
 else
  m(250, 0);
 pln;
 for i := 1 to z.p do
  for k := 1 to j do
   gm[i, k] := 0;
end;
procedure gr;
 var
  i, j, ct, st : w;
  sr, zst : str10;
  a : str256;
begin
 pln;
 m(220, 0);
 pln;
 if z.j <> 0 then
  begin
   ct := 0;
   st := z.mf + z.mm + 1;
   sr := z.na[st];
   while (ct <= z.j) and (st <= z.mf + z.mm + z.j) do
    begin
     zst := z.na[st];
     m(193, 0);
```

```
      write(y, zst, '  :   ');
      write(y, '   (   ');
      j := 1;
      while (sr = zst) do
       begin
        write(y, '                    , ');
        ba(a, z.kn[st], 1);
        for i := 1 to z.p do
         if bt(z.kn[st], i) then
          gm[i, j] := 1
         else
          gm[i, j] := 0;
        st := st + 1;
        ct := ct + 1;
        if (st <= z.mf + z.mm + z.mj) then
         sr := z.na[st];
        j := j + 1;
       end;
      sr := z.na[st];
      zst := z.na[st];
      write(y, '                              ) ');
      wy;
      gr2(j);
     end;
   end;
end;
procedure dv (var a : q;
         var b : q);
 var
  i : w;
  c, d : q;
  ok : bl;
begin
 i := 1;
 ok := ff;
 while (not ok) and (i <= mvd) do
  begin
   if il(a, ip[i]) and il(a, op[i]) then
    if not triv(ip[i], op[i], a) and not key(ip[i], a) then
     begin
      uo(ip[i], op[i], b);
```

```
        ms(a, b, c);
        uo(c, ip[i], a);
        ok := tt;
       end;
    i := i + 1;
  end;
end;
procedure gen4nf;
 var
  f1, f2, ok : bl;
   sc, rsc : q;
begin
 sc := uni;
 rsc := 0;
 pln;
 m(226, 0);
 pln;
 f1 := ff;
 f2 := ff;
 ok := ff;
 while not ok do
  begin
   dv(sc, rsc);
   if tst4nf(sc) then
    begin
     f1 := tt;
     ba(att, sc, 1);
    end;
   if tst4nf(rsc) then
    begin
     f2 := tt;
     ba(att, rsc, 1);
    end;
   if f1 and f2 then
    ok := tt;
   if f1 then
    sc := rsc;
  end;
 pln;
end;
procedure handlemenu;
```

```
var
  i, j : w;
  a : str256;
begin
 case WInLineF($A86A, nm) of
 100 :
   case WInLineF($A86B, nm) of
    1 :
     hideall;
    2 :
     pdb;
    3 :
     begin
      M(79, 1);
      readln(att);
      sv := ff;
      if att = 'y' then
       sv := tt;
      M(82, 1);
      readln(att);
      done := tt;
      if att = 'y' then
       nomore := tt;
     end;
    4 :
     begin
      M(1, 1);
      readln(n);
      case n of
       1 :
        M(7, 1);
       2 :
        M(28, 1);
       3 :
        m(34, 1);
       otherwise
        M(100, 1);
      end;
     end;
   end;
  102 :
```

```
case WInLineF($A86B, nm) of
 1 :
  begin
   M(49, 1);
   readln(att);
   M(91, 0);
   writeln(y, att);
   M(175, 0);
   impl;
   pln;
  end;
 2 :
  begin
   M(55, 1);
   M(85, 0);
   readln(att);
   writeln(y, att);
   M(175, 0);
   ab(att, cv);
   db(x, cv);
   if (z.m = 0) and (z.j = 0) then
    clos := cl(cv, 0)
   else
    clos := cl(cv, 1);
   ba(att, clos, 0);
   ba(att, clos, 1);
   pln;
   M(88, 0);
   M(175, 0);
   for i := 1 to x do
    for j := 1 to z.p do
     if bt(dt[i], j) then
      z.token[-j, 1] := i;
   for i := 1 to x do
    begin
     write(y, '> ');
     ba(a, dt[i], 1);
     ba(a, dt[i], 0);
    end;
   pln;
   cln;
```

```
 end;
3 :
 begin
  m(232, 1);
  readln(n);
  case n of
   1 :
    begin
     m(70, 1);
     readln(att);
     if att = 'universe' then
      cv := uni
     else
      ab(att, cv);
     m(73, 1);
     readln(at1);
     ab(at1, cv1);
     writeln(y, 'Testing whether   (', at1, ')   is a key of
          the set of attributes (', att, '  )');
     if key(cv1, cv) then
      m(169, 0)
     else
      m(172, 0);
     cln;
     pln;
    end;
   2 :
    begin
     m(70, 1);
     readln(att);
     if att = 'universe' then
      cv := uni
     else
      ab(att, cv);
     m(67, 0);
     writeln(y, att);
     m(175, 0);
     fkey(cv, cv1);
     cln;
     pln;
    end;
```

```
    end;
  end;
  4 :
   begin
    M(13, 1);
    readln(n);
    case n of
     1 :
      m(22, 1);
     2 :
      M(16, 1);
     3 :
      M(10, 1);
     4 :
      M(25, 1);
     otherwise
      M(103, 1);
    end;
   end;
 end;
101 :
 case WInLineF($A86B, nm) of
  1 :
   gen3nf;
  2 :
   ;
  3 :
   genbcnf;
  4 :
   begin
    m(70, 1);
    m(199, 0);
    readln(att);
    writeln(y, att);
    ab(att, cv);
    if tst4nf(cv) then
     m(169, 0)
    else
     m(172, 0);
    pln;
   end;
```

```
5 :
 gen4nf;
6 :
 gr;
7 :
 begin
  m(202, 1);
  readln(n);
  case n of
   1 :
    m(214, 1);
   2 :
    m(235, 1);
   3 :
    m(208, 1);
   4 :
    m(217, 1);
   5 :
    m(211, 1);
   6 :
    m(244, 1);
   otherwise
    m(205, 1);
  end;
 end;
end;
103 :
 case WInLineF($A86B, nm) of
  1 :
   ;
  2 :
   ;
  3 :
   ;
  4 :
   begin
    m(37, 1);
    readln(n);
    case n of
     1 :
      m(40, 1);
```

```
            2 :
              m(43, 1);
            3 :
              m(61, 1);
            otherwise
              m(109, 1);
           end;
         end;
       end;
      otherwise
       ;
    end;
  end;(*proc*)
begin
  rewrite(y, 'printer:');
  write(y, chr(27), 'c', chr(27), 'p');
  write(y, chr(29), 'A@');
  for delay := 1 to 58 do
   write(y, '@@');
  write(y, 'C@@@@@@@@@@@@@@A@', chr(30));
  psz := 10;
  tsz := 8;
  nomore := ff;
  nt := ff;
  fir := ff;
  rec := ff;
  om := 999999;
  scr('WELCOME TO  G L O R D ');
  for delay := 1 to 3000 do
    ;
  init;
  hideall;
  while nomore = ff do
   begin
     done := ff;
     open(msgvar, 'msgsII');
     mt := msgvar^;
     close(msgvar);
     pmp := ' database file :';
     dbfile := oldfilename(pmp);
     open(dbvar, dbfile);
```

```
z := dbvar^;
close(dbvar);
tr;
pln;
M(178, 0);
pln;
wy;
writeln(y, 'Given: the database is:   ', dbfile);
pln;
uni := 0;
ul := 0;
for n := 1 to z.p do
 sb(uni, n, tt);
for n := 1 to z.t do
 for dy := 1 to z.p do
  if bt(z.kn[n], dy) then
   sb(ul, dy, tt)
  else if bt(z.ou[n], dy) then
   sb(ul, dy, tt);
repeat
 if testmouse(ib) then
  begin
   if ib then
    begin
     fir := tt;
     om := LinLineF($A93D, ev.wr);
    end;
   nm := om;
   Handlemenu;
  end;
until done;
M(181, 0);
pln;
if sv then
 begin
  open(dbvar, dbfile);
  dbvar^ := z;
  put(dbvar);
  close(dbvar);
 end;
end;
```

```
  close(y);
  InlineP($A932, sm);
  InLineP($A93C, OldMenuBar);
  InLineP($A937);
  scr('              BYE');
end.
```

REFERENCES

AHO79 Aho, A.V., Beeri, C. and Ullman, J.D., "The theory of joins in relational databases", ACM Trans. on Database Systems, Vol.4, No.3, (Sept. 1979), pp. 297-314.

ARMS74 Armstrong, W.W., "Dependency structure of database relationships", Proc. IFIP 74, Geneva, Switzerland, (1974), pp. 580-583.

AROR81 Arora, S.K. and Smith, K.C., "Graphical normal forms based on root dependencies in relational data base systems", Inter. J. of Computer and Information Sciences, Vol.10, No.4, (1981); pp. 235-259.

AUSI81 Ausiello, G., D'Atri, A. and Sacca, D., "Graph algorithms for the synthesis and manipulation of data base schemes", Proc. on Graph Theoretic Concepts in Computer Science, Lecture Notes in Computer Science, Vol.100, Springer-Verlag, (1981), pp. 212-233.

AUSI83 Ausiello, G., D'Atri, A. and Sacca, D., "Graph algorithms for functional dependency manipulation", J. ACM, Vol.30, No.4, (Oct. 1983), pp. 752-766.

BEER77 Beeri, C., Fagin, R. and Howard, J., "A complete axiomatization for functional and multivalued dependencies in database relations", IBM Research Report RJ1977, IBM Research Lab., San Jose, CA, (Jan. 1977).

BEER79 Beeri, C. and Bernstein, P.A., "Computational problems related to the design of normal form relational schemas", ACM Trans. on Database Systems, Vol.4, No.1, (March 1979), pp. 30-59.

BEER80a Beeri, C. and Vardi, M.Y., "The implication problem for data dependencies", Proc. 8th. ICALP, Acre Israel, 1981, in Lecture Notes in Computer Science, Vol. 115, (1981), pp. 73-85.

BEER80b Beeri, C., "On the membership problem for functional and multivalued dependencies in relational databases", ACM Trans. on Database Systems, Vol.5, No.3, (Sept. 1980), pp. 241-259.

BEER81c Beeri, C. and Vardi, M.Y., "The implication problem for data dependencies", Proc. 8th Colloquium on Automata, Languages and Programming ACTC (AKKO). Springer-Verlag, Berlin and New York, (1981), pp. 73-85.

BERN76 Bernstein, P.A., "Synthesizing third normal form relations from functional dependencies", ACM Trans. on Database Systems, Vol.1, No.4, (Dec. 1976), pp. 277-298.

BJOR83 Bjornstedt, H. and Hulten, C., "RED1, a database design tool for the relational model", SYSLAB report No. 18, (March 1983), University of Stocholm, Sweeden.

BJOR84 Bjornstedt, H., "RED1, A database design tool for the relational model of data", Database Engineering, Vol.7, No.4, (1984), pp. 34-39.

CASA84 Casanova, M.A., Fagin, R. and Papdamitriou, C.H., "Inclusion dependencies and their interaction with functional dependencies", J. of Computer and System Sciences, Vol.28, (1984), pp. 29-59.

CERI86 Ceri, S. and Gottlob, G., "Normalization of relations and Prolog", CACM, Vol.29, No.6, (June 1986), pp. 524-544.

CHAN81 Chandra, A.K., Lewis, H.R. and Makowsky, J.A., "Embedded implicational dependencies and their inference problems", ACM Symposium on Theory of Computing, Milwaukee, (1981), pp. 342-354.

CHEU86 Cheung, T-Y., "A Petri-Net model for functional dependencies in relational databases", TR-85-07, Department of Computer Science, University of Ottawa, Ottawa, Ontario, Canada.

CHEU86 Cheung, T-Y., "A Petri-net-based graphical model for logical relational database design", Proc. Inter. Computer Symposium, Tainan, Taiwan, (1986).

CODD72 Codd, "Further normalization of the database relational model", Database systems (R.Rustin, ed.), Printice-Hall, Englewood cliffs, New Jersey, (1972), pp. 33-64.

CODD74 Codd, E.F., "Recent investigations in relational database systems", Proc. IFIP Conf., (1974), pp. 1012-1021.

CODD75 Codd, E.F., "Understanding relations", FDT 7:3-4, ACM, New-York, (1975), pp. 23-28.

DAHL82 Dahl, V., "On database systems development through logic", ACM Trans. on Database Systems, Vol. 7, No.1, (Mar. 1982), pp. 102-123.

DATE85 Date, C.J. Introduction to Database Systems, Vol.1 (fourth edition),Addison-Wesley, (1985).

FAGI77a Fagin, R., "Multivalued dependencies and a new normal form for relational databases", ACM Trans. on Database Systems, Vol.2, No.3, (Sept. 1977), pp. 262-278.

FAGI77b Fagin, R., "Functional dependencies in a relational database and propositional logic", IBM J. of Res. and Devel., Vol.21, No.6, (1977), pp. 534-544.

FAGI80 Fagin, R., "Horn clauses and database dependencies", J. ACM, Vol.29, No.4, (Oct. 1982), pp. 952-985.

FISH83 Fisher, P.C. and Tsou, D.M., "Whether a set of multivalued dependencies implies a join dependency is NP-hard", SIAM J. of Computing, Vol.12, No.2, (May 1983), pp. 259-266.

GALI82 Galil, Z., "An almost linear-time algorithm for computing the dependency basis in a relational database", J. ACM, Vol.29, No.1, (Jan. 1982), pp. 96-102.

GALL78 Gallaire, H. and Minker, J., Logic and Databases, Plenum, New York, (1978).

GALL81a Gallaire, H. and Nicholas, J.-M., Eds., Advances in Data Base Theory, 1981, Vol. 1, Plenum, New York, (1981).

GALL81b Gallaire, H., "Impacts of logic on databases", Proc. 7th Inter. Conf. on Very Large Data Bases, France, (Sept. 1981), pp. 248-259.

GALL83 Gallaire, H., "Logic databases vs. deductive databases", Logic Programming Workshop, Portugal, (1983), pp. 608-622.

GALL84a Gallaire, H., Minker, J. and Nicholas, J.-M., Eds., Advances in Data Base Theory, Vol. 3, Plenum, New York, (1984).

GALL84b Gallaire, H., Minker, J. and Nicholas, J.-M., "Logic and databases: a deductive approach", ACM Computing Surveys, Vol.16, No.2, (Jun.1984), pp. 153-185.

GINS83 Ginsburg, S. and Hull, R., "Chracterizations for functional dependency and Boyce-Codd normal form families", Theoretical Computer Science, Vol.26, (Sept. 1983), pp. 243-286.

HAGI79 Hagihara, K., Ito, M., Tanigushi, K. and Kasami, T., "Decision problems for multivalued dependencies in relational databases", SIAM J. on Computing, Vol.8, No.2, (May.1979), pp. 247-264.

HARE79 Harel, D., "Review of logic and databases", ACM Computing Review, Vol. 21, No.8, (Aug. 1980), pp. 367-369.

INRI83 INRIA Institut National de Recherche en Informatique et en Automatique, "Bases de donnees: Nouvelles perspectives", rapport du groupe BD3, (Jan. 1983).

JACO82 Jacobs, B.E., "On database logic", J. ACM, Vol. 29, No.2, (Apr.1982), pp. 310-332.

KOWA79 Kowalski, R.A., Logic for Problem Solving. Elsevier North-Holland, New York, (1979).

KUNI82 Kunifuji, S. and Yokota, H., PROLOG and relational databases for the fifth generation computer system. ICOT Rep. D02, Institute for New Generation Computer Technology, Tokyo, (1982).

MAIE79 Maier, D., Mendelzon, A.O. and Sagiv, Y., "Testing implications of data dependencies", ACM Trans. on Database Systems, Vol.4, No.4, (Dec. 1979), pp. 455-469.

MAIE80 Maier, D., "Minimum covers in the relational database model", J. ACM, Vol.27, No.4, (Oct. 1980), pp. 664-674.

MAIE81 Maier, D., Sagiv, Y. and Yannakakis, M., "On the complexity of testing implications of functional and join dependencies", J. ACM, Vol.28, No.4, (Oct. 1981), pp. 680-695.

MAIE83 Maier, D., The Theory of Relational Databases, Computer Science Press, (1983).

MATW85 Matwin, S. and Pietrzykowski. T., "PROGRAPH: a preliminary report", Computer languages, Vol.10, No.2, (1985), pp. 91-126.

MEND79 Mendelżon, A.o., "On axiomatizing multivalued dependencies in relational databases", J. ACM, Vol.26, No.1, (Jan. 1979), pp. 37-44.

MINK78 Minker, J., "An experimental relational database system based on logic", In Logic and Databases, Plenum, New York, (1978), pp. 107-147.

NICH78a Nicholas, J.-M, "First order logic formalization for functional, multivalued and mutual dependencies", Proc. ACM-SIGMOD Inter. Conf. on management of data, New York, (June 1978), pp. 40-46.

NICH78b Nicholas, J.-M. and Gallaire, H., "Database: Theory vs Interpretation", Logic and Databases, Plenum, New York, (1978), pp. 33-54.

NICH78c Nicholas, J.-M. and Yazdanian, K., "Integrity checking in deductive databases", Logic and Databases, Plenum, New York, (1978), pp. 325-346.

MITCH83 Mitchell, J.C., "Inference rules for functional and inclusion dependencies", Proc. 2nd ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, (March 1983), pp. 58-69.

NAMB83 Nambiar, Radhakrishnan, T. and Tikekar, V.G., "Representation of functional dependencies in relation databases using linear graphs", Theoretical Computer Science, North Holland, Vol.24, (July 1983), pp. 143-160.

PETE79 Peterson, J.L., "Petri-nets", Computing Surveys, Vol.9, (Sept.1977), pp. 223-252.

RISS79 Rissanen, J., "Theory of relations for databases - a tutorial survey", Proc. 7th Symp. on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, Vol.64, Springer-Verlag, Berlin, (1979), pp. 537-551.

SADR80a Sadri, F. and Ullman, J.D., "A complete axiomatiation for a large class of dependencies in relational databases", J. of ACM, Vol. 27, No.1, (1980), pp. 117-122.

SADR80b Sadri, F., and Ullman, J.D., "The interaction between functional dependencies and template dependencies", ACM SIGMOD Conf., (1980), pp. 45-51.

SADR82a Sadri, Y. and Ullman, J.D., "The theory of functional and template dependencies", Theor. Comput. Sci., Vol. 17, (1982), pp. 317-332.

SADR82b Sadri, Y. and Ullman, J.D., "Template dependencies: A large class of dependencies in relational databases and its complete axiomatization", J. ACM, Vol.29, No. 2, (Apr. 1982), pp. 363-372.

SAGI79 Sagiv, Y. and Fagin, R., "An equivalence between relational database dependencies and a subclass of propositional logic", IBM Research Report RJ2500, IBM Research Lab., San Jose, CA, (1979).

SAGI80 Sagiv, Y., "An algorithm for inferring multivalued dependencies with an application to propositional logic", J. ACM. Vol 27, No. 2, (Apr. 1980), pp. 250-262.

SAGI81 Sagiv, Y., Delobel, C., Parker, D.S. and Fagin, R., "An equivalence between relational database dependencies and a subclass of propositional logic", J. ACM, Vol.28, No. 3, (July 1981), pp. 435-453.

SCIO83a Sciore, E., "Inference rules for functional and inclusion dependencies", Proc. 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, (March 1983), pp. 48-57.

SCIO83b Sciore, E., "Improving database schemes by adding attributes", ACM Trans. on Database Systems, (1983), pp. 379-383.

TAMA83 Tamassia, R., Batini, C. and Talamo, M., Proc. 3rd Inter. Conf. on Entity-Relationship approach to software engineering, Ed. C.G. Davis et al., Anaheim, CA, (Oct. 1983), pp. 421-439.

ULLM82 Ullman, J.D., Principles of Database Systems, Computer Science Press, Potomac, Maryland, (1982).

ULLM85 Ullman, J.D., "Implementation of logical query languages for databases", ACM Trans. on Database Systems, Vol.10, No.3 (Sept 1985), pp. 289-321.

VARD80 Vardi, M.Y., "Inferring multivalued dependencies from functional and join dependencies", Acta Informatica, Vol. 19, No. 1, (1983), pp. 305-324.

YANN80 Yannakakis, M. and Papadimitriou, C.H., "Algebraic dependencies", Proc. 21st Annual IEEE Symp. on Foundations of Computer Science,(1980), pp. 328-332.

YANN81 Yannakakis, M., "Algorithms for acyclic database schemes", Proc. 7th Inter. Conf. on Very Large Data Bases, Cannes, (Sept. 1981), pp. 82-94.

ZANI76 Zaniolo, C., "Analysis and design of relational schemata for database systems", Doctoral dissertation, UCLA, (July 1976).

ZANI81 Zaniolo, C. and Melkanoff, M.A., "On the design of relational database schemata", ACM TODS, Vol.6, No.1, (March 1981), pp. 1-47.

ZANI82 Zaniolo, C. and Melkanoff, M.A., "A formal approach to the definition and the design of conceptual schemata for database systems", ACM TODS, Vol.7, No.1, (March 1982), pp. 24-59.

ZHAN83 Zhang, Z., Mendelzon and Alberto, O., "A graphical query language for entity relationship databases", Proc. 3rd Inter. Conf. on Entity-Relationship Approach to software Engineering, Ed. C.G. Davis et al., Anaheim, CA, (Oct.83), pp. 441-448.

ZHU84 Zhu, Y., "Line graph of gamma-acyclic database schemes and its recognition", Proc. 10th Conf. on VLDB, Singapore, (1984), pp. 218-221.