tanulmányok 133/1982

MTA Számitástechnikai és Automatizálási Kutató Intézet Budapest

MAGYAR TUDOMÁNYOS AKADÉMIA
SZÁMITÁSTECHNIKAI ÉS AUTOMATIZÁLÁSI KUTATO INTÉZETE

РГ-II / КПВВТ

"СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ И ИНФОРМАЦИОННЫЕ СИСТЕМЫ"

С Б О Р Н И К

НАУЧНО-ИССЛЕДОВАТЕЛЬСКИХ РАБОТ

ТОМ III

Tanulmányok 133/1982

A kiadásért felelős:

DR VÁMOS TIBOR

ISBN 963 311 138 2

ISSN 0324 2951

Főprint nyomda 82029

С О Д Е Р Ж А Н И Е

Сборник научно-исследовательских работ
рабочей группы РГ-II, КНВВТ

выпуск I3I - том I.

выпуск 132 - том II

выпуск 133 - том III

MTA Számitástechnikai és Automatizálási Kutató Intézete, Tanulmányok 133/1982 Proc. of RG-11, KNVVT

A LOCAL INFORMATION STATION - VERSION ONE

P.Barnev, At.Radensky, P.Azalov, Kr.Markov, Z.Vassilev

Institute of Mathematics with Computer Center
Bulgaria 1090-Sofia, P.O.Box 373

ABSTRACT

The paper is an introduction to the command language of the Local Information Station - 1 database system. An extensive set of examples is used to describe the action of each command. Appendix 1 contains a list of all system commands.

0. Introduction

The family of local information stations (LIST) is based on the concept of information servising, developed by P. Barnev and collaborators at the Institute of Mathematics of the Bulgarian Academy of Sciences /1-9/.

LIST-1 is the smallest and simplest for use system of the LIST family. It is designed to manage a tabular set of data for a single user.

LIST-1 features simple use and a powerful set of database operations.

LIST-1:

- runs on a SM-4 minicomputer;
- creates, accesses and maintains the user's database;
- interacts with the user via a terminal using a simple command language;
- allows the advanced user to define new commands as

sequences of commands as a tool for system adaptation to the specific user requirements.

1. Primaries

1.1. An exemplary database

Throughout the command descriptions we are going to use an example of a **simple** database consisting of 2 tables, or relations:

- a relation named 'theaters' with attributes (columns) 'name', 'head' and 'year of foundation',
- a relation named 'actors' with attributes 'actor name' and 'theater'.

NAME	HEAD	YEAR OF FOUNDATION
National theater	Dimov	1890
Drama theater	Raev	1902
Comedy theater	Michailov	1960
Theater of the smile	Ivanov	1927
Puppet show theater	Tassev	1934
Ballet and show theater	Malchev	1893
Army Theater	Penchev	1948

Example 1.1.1 Relation 'theaters'.

ACTOR NAME	THEATER
Michailov	Comedy theater
Kolev	Army theater
Markov	Theater of the Army
Panova	Theater of the smile
Dimov	National theater
Goranova	Drama theater
Jekova	Show theater

Example 1.1.2 Relation 'actors'.

1.2. The database structure

The user views the database as a collection of relations (tables with fixed number of columns and variable number of rows). Each individual fact is represented in the database as a row (tuple) in a relation, consisting of as many elementary data items as are the attributes (columns) of the relation. A relation may have zero or more tuples and at least one attribute. Names are used to refer to both relations and attributes.

All elementary data items are character strings of any length. In case a string has several leading digits it is considered a numeric data item which evaluates to the decimal number,represented by the leading digits of the string.

$28.5 -13.988 .57 98% 147CR @456.70

Example 1.2.1 Numeric data items.

All other elementary data items will be called non-numeric or text ones.

E-10 London W1 TEXTtext POB 373 Sofia1000

Example 1.2.2 Text data items.

1.3. Comparing elementary data items

Numeric data items can be tested for conditions such as equality (=), not-equality (/=), less than (<), less or equal (<=), greater than (>) and greater or equal (>=).

98% $\geq$ 97tons, 147 parts = 147 cows, $9.00 = £9.

Example 1.3.1 All tests evaluate to 'true'.

Non-numeric data items can be tested for the same conditions. In the lexicographic order of characters the let-

ters follow the blank character, the digits and the special characters.

Dimov < Roev, Mark < Markov, LIST-1 = list-1

Example 1.3.2 All tests evaluate to 'true'.

When comparing two data items, LIST-1 attempts to treat them as numeric and compare their values. In case of failure character string comparison is performed.

1.4. Pattern matching

LIST-1 provides for the matching of a data item, treated as a character string, against a pattern. The operation is denoted '?'.

The pattern is composed of character string and several special characters - '*','&','|','/'. A data item matches a character string pattern iff they are equal. A data item matches the concatenation of two patterns iff it can be subdivided into two strings which match the first and second pattern strings respectively. Any data item matches the special pattern '*'.

Theater of the army ? T*y
Theater ? *t*e*
Army Theater ? *T*a*t*

Example 1.4.1 All the above matches evaluate to 'true'.

Composite patterns may be constructed using pattern concatenation and the logical connectors and (&), or (|) and not (/). Formally, a string matches the pattern P1 & P2 iff it matches both P1 and P2. A string matches P1 | P2 iff it matches either P1 or P2. A string matches / P1 iff it does not match P1.

Army theater ? Army* & *theater
Army theater ? *theater* | *army
Comedy theater ? *theater* & C*

Example 1.4.2 All tests evaluate to 'true'.

D & V
*D & *V
D & / *D*

Example 1.4.3 The above patterns will not match any string.

1.5. Querying the database

To access the elementary data items stored in a LIST-1 database the user has to identify the relation, tuple and attribute of the specific data item. Relations and attributes are referred to by their names. The specific tuple is identified by specifying a condition on the values of its attributes. The condition may be a simple comparison or pattern match between attribute values or an attribute value and an elementary data item (constant). In the former case, when attributes of different relations are involved, the only comparison allowable is equality and the operation is denotes ':=:'. Composite conditions may be constructed by using the logical connectors &, | , and /. Parentheses are also allowed.

A condition identifies all tuples of the cartesian product of the relations concerned, for whose data items the condition test evaluates to 'true'.

(head = Penchev) & (name ? *Army*)
(name ? (A*t* | T*army))
(year of foundation>= 1900 & year of foundation<=1940)
(head :=: actor name)

1.5.1 All conditions are valid for LIST-1. All but the last are over the 'theaters' relation and the last one concerns both relations of the exemplary database.

1.6. Communicating with LIST-1.

The interaction between a user and LIST-1 is done via a terminal. The user specifies his intentions by issuing LIST-1 commands. The system can help the user by prompting for additional information it needs and by sending warning and error messages. A 'help' facility is also available to give the user all necessary information on system commands.

The inexperienced user may leave the initiative completely to LIST-1 by specifying only the command codes. LIST-1 will question him for all additional information it needs in an easy to understand manner.

1.7. LIST-1 commands

The commands of LIST-1 comprise three principal groups: basic commands, macro facility commands and service commands.

2. Basic commands of LIST-1.

2.1. Creation of tables and insertion of tuples. Here are some examples of these commands:

```
ENTER COMMAND:
121
ENTER TABLE NAME:
theaters
ENTER COLUMN NAMES:
name,head,year of foundation
ENTER COMMAND:
```

Example 2.1.1 Creation of table 'theaters'.

```
ENTER COMMAND:
121
ENTER TABLE NAME:
actors
ENTER COLUMN NAMES:
actor name, theater
ENTER COMMAND:
```

Example 2.1.2 Creation of table 'actors'.

```
ENTER COMMAND:
130
ENTER TABLE NAME:
theaters
NAME:
National theater
HEAD:
Dimov
YEAR OF FOUNDATION:
1890
TUPLE ENTERED:
130
NAME:
Drama theater
HEAD:
Raev
YEAR OF FOUNDATION:
1902
TUPLE ENTERED:
013
ENTER COMMAND:
```

Example 2.1.3 Entering 2 tuples in the 'theaters' relation.

In order to create a table, the user has to specify its name and the names of all attributes as parameters of command 121. Command 130 is used to enter (insert) tuples in an existing relation.

The experienced user can follow the 'fast path' by issuing a 230 command instead of a 130.

```
230 - table (theaters)
    - columns (name, year of foundation, head)
    - tuples (National theater, 1890, Dimov;
              Drama theater, 1902, Raev;
              Comedy theater, 1960, Michailov)
```

Example 2.1.4 Using command 230 to enter tuples in the 'theaters' relation.

In case the database contains two relations with similar attributes, the relations may be linked together by inserting the tuples of one of them into the other (command 231).

2.2. Deletion from the database

Here is the next example:

```
ENTER COMMAND:
160
ENTER TABLE NAME:
actors
ACTOR NAME:
Jekova
THEATER:
*show*
TUPLE DELETED:
013
ENTER COMMAND:
```

Example 2.2.1 Deleting tuples from a relation.

```
ENTER COMMAND:
160
ENTER TABLE NAME:
theaters
NAME:

YEAR OF FOUNDATION:

TUPLES DELETED:
013
ENTER COMMAND:
```

Example 2.2.2 Deleting all tuples of a relation.

```
ENTER COMMAND:
151
ENTER TABLE NAME:
theaters
TABLE DELETED:
ENTER COMMAND:
```

Example 2.2.3 Deleting a table from the database.

The above examples illustrate the use of commands 160 and 151 - tuple and table deletion. When promted with a column name in the 160-command intercourse the user is expected to specify a simple condition on the value of the co - lumn. The default condition is ' ? * ' (see Example 2.2.2).

2.3 Update commands

The LIST-1 tuple update commands will be illustrated by the following examples:

```
ENTER COMMAND:
170
ENTER TABLE NAME:
theaters
NAME:
```

```
Ballet and show theater
HEAD:

YEAR OF FOUNDATION:

UPDATES:
NAME:BALLET AND SHOW THEATER

HEAD: MALCHEV
Kalchev
YEAR OF FOUNDATION: 1893
/0
TUPLE UPDATED:
ENTER COMMAND:
```

Example 2.3.1 Updating a tuple.

```
ENTER COMMAND:
270 - table (theaters)
    - condition (name = Ballet and show theater)
    - columns (head, year of foundation)
    - tuples (Kalchev,/0)
TUPLES UPDATED:
ENTER COMMAND:
```

Example 2.3.2 Update a tuple - the fast path.

To update the elementary data items in a relation one uses the 170 LIST command (see Example 2.3.1) or the 270 command (see Example 2.3.2). The data item value '/0' is a notation for the 'unde-fined' value.

2.4 Information retrieval

We begin with examples again:

```
ENTER COMMAND:
144
ENTER TABLE NAME:
theaters
NAME:

HEAD:

YEAR OF FOUNDATION:
 ∠ = 1900
THEATERS.
NAME                          HEAD    YEAR OF FOUNDATION
NATIONAL THEATER              DIMOV       1890
BALLET AND SHOW THEATER       MALCHEV     1893
ENTER COMMAND:
```

Example 2.4.1 Retrieving tuples from a LIST-1 relation.

```
ENTER COMMAND:
244 - table (theaters)
    - columns (actor name)
    - condition (theater  ?  ( A * r | * army *) )
ACTORS.
ACTOR NAME
KOLEV
MARKOV
ENTER COMMAND:
```

Example 2.4.2 Retrieving specified columns of a relation - the fast path.

All query commands require the specification of the relation (table) name, the requested attributes' (columns) names and a condition identifying the tuples to be retrieved. If the condition test evaluates 'true' for all tuples of the

relation, all tuples will be displayed. Two-table queries are also possible.

```
ENTER COMMAND:
243 - tables (theaters, actors)
    - columns (head)
    - condition (head :=: actor name)
HEAD
MICHAILOV
DIMOV
ENTER COMMAND:
```

Example 2.4.3 Retrieving the names of all actors that are theater heads.

Such queries may be viewed as ranging over the cartesian product of the relations involved.

3. Macro facility commands

LIST-1 allows for the definition of new commands as sequences of system commands. The user-defined commands will be called 'procedures'.

```
ENTER COMMAND:
200 - name (create)
    - parameters (tab,col)
    - body (121 - table (tab) - columns (col) )
ENTER COMMAND:
create - tab(actors) - col(actor name, theater)
```

Example 3.1 Definition and usage of a 'create' command.

The parameter passing is done by direct replacement of character strings in the procedure body.

The command 201 is used to delete a user defined procedure.

```
ENTER COMMAND:
201 - name (create)
PROCEDURE DELETED:
ENTER COMMAND:
```

Example 3.2 Procedure deletion.

Default parameter values may also be defined in a procedure definition. A procedure may invoke other command procedures, but no recursion is allowed.

4. Service commands

4.1. Indexing

To decrease search time LIST-1 provides for a special mechanism, called 'index', on columns of tables that participate in the search argument. The index, if present, will automatically be used to speed up all searches involving the indexed column (attribute). When created or destroyed indices are identified by names. The user can define indices where the values of the column are ordered either up or down, thus defining the order of tuples in all retrieve or update operations.

```
ENTER COMMAND:
125
ENTER INDEX NAME:
years
ENTER TABLE NAME:
theaters
ENTER COLUMN NAME:
year of foundation
INDEX CREATED:
ENTER COMMAND:
```

Example 4.1.1 Index creation.

```
ENTER COMMAND:
155
ENTER INDEX NAME:
years
INDEX DESTROYED:
ENTER COMMAND:
```

Example 4.1.2 Index destroyal.

4.2. Getting general information about the database

The user may request information about the tables in the database (command 141) and their columns (command 142). The number of tuples in a relation whose attribute values fulfill a given condition may also be displayed (246 and 247 commands).

```
ENTER COMMAND:
142
ENTER TABLE NAME:
theaters
COLUMNS OF TABLE THEATERS:
HEAD,NAME,YEAR OF FOUNDATION
INDICES OVER COLUMNS OF TABLE THEATERS:
YEARS          OVER YEAR OF FOUNDATION
ENTER COMMAND:
```

Example 4.2.1 Displaying information about a table in the database.

4.3 Messages

The advanced user may want to change the LIST-1 system messages. LIST-1 provides 2 commands - 280 and 281 for sending user messages to the terminal and changing system messages respectively.

```
ENTER COMMAND:
200 - name (русский)
    - body (281 - oldmsg (ENTER COMMAND:)
                - newmsg (ЗАДАЙ КОМАНДУ:)
           280 - msg (ДАЛЬШЕ ДИАЛОГ БУДЕТ НА
                      РУССКОМ ЯЗЫКЕ;      ))

ENTER COMMAND:
русский
ДАЛЬШЕ ДИАЛОГ БУДЕТ НА РУССКОМ ЯЗЫКЕ;
ЗАДАЙ КОМАНДУ:
```

Example 4.3.1 System message change.

4.4. Interfacing to the operating system

LIST-1 allows for directly sending commands to the operating system (command 222). A LIST-1 command file execution is also possible by using command 221. Command 220 is used to redirect the LIST-1 output to a file, instead of the user's terminal.

Activation and deactivation of LIST-1 depend on the specific operating system. Command execution may be terminated by responding '013' to any LIST-1 question.

5. Concluding remarks

The main requirements to the implementation of LIST-1 are:

- portability;
- small main memory requirements;
- reliability and simple use.

Portability is achieved by using FORTRAN as the implementation language and by programming LIST-1 as a complex of small modules. System modularity also keeps the main storage requirements relatively small. Reliability is achieved

by using proper protection machanisms.

The LIST-1 database is implemented as a complex linked list structure - a decision imposed by the requirement that an elementary data item may be a string of any length.

Appendix 1
LIST-1 command syntax

1. Basic commands
1.1. Table creation commands
1.1.1. Including a table in the database
121 - table (table name)
- columns (column name [,... column name n])
1.1.2. Inserting tuples in a relation
230 - table (table name)
- columns (column name 1 [,... column name n])
- tuples (column item 1 [,...column item n]
[;... column item 1m [,... column item nm]])
130
1.1.3. Combining two tables
231 - object (table name)
- source (table name)
1.2. Deletion commands
1.2.1. Table deletion
151 - table (table name)
1.2.2. Deletion of tuples from a table
260 - table (table name)
- condition (condition expression)
160
1.3. Update commands
1.3.1. Updating tuples in a table
170
270 - table (table name)
- condition (condition expression)

- columns (column name 1 [,...column name n])
- tuples (data item [,...data item n] [;...
  data item 1m [,...data item nm]])

1.4. Information retrieval commands

1.4.1. Tuple retrieval from a single table

244 - table (table name)
- columns (column name 1 [,...column name n])
- condition (condition expression)

144

1.4.2. Tuple retrieval from 2 tables

243 - tables (table name 1,table name 2)
- columns (column name 1 [,... column name n])
- condition (condition expression)

2. Macro facility commands

2.1. Definition of a command procedure

200 - name (procedure name)
- parameters (param1 [,... param n])
- body (command1 [,... command m])

2.2. Deleting a command procedure

201 - name (procedure name)

3. Service commands

3.1. Indices

3.1.1. Index creation

126 - index (index name)
- table (table name)
- column (column name)

127 - index (index name)
- table (table name)
- column (column name)

3.1.2. Index destroyal

155 - index (index name)

3.2. Getting statistical information

3.2.1. Database tables list

141

3.2.2. Table description

142 - table (table name)

3.2.3. Number of tuples in a table (or cartesian product of tables)

246 - tables (table name 1, table name 2)
- condition (condition expression)

247 - table (table name)
- condition (condition expression)

3.3. Messages

3.3.1. Sending a message to the user's terminal

280 -msg (message text)

3.3.2. Change of a system message

281 - oldmsg (message text 1)
- newmsg (message text 2)

3.4. Interface to the operating system

3.4.1. Redirecting the system output to a file

220 - command (command text)
- file (output file name)

3.4.2. Command file execution

221 - file (command file name)

References

1. P. Barnev, Systéme abstract de service informatique des collectivites, Congres de mathematique appliquees, Thessalonique, 1976.
2. P. Barnev, Systems for information servicing of collectivities, Serdica, vol. 4, fasc 2-3, 1978.
3. P. Barnev, At. Radenski, Structure of the information and operations on the entities in a system for information servicing of collectivities, Serdica, vol. 4, fasc 2-3, 1978.

4. **K. Ivanov, Concretization of the collectivity and individualizing the communication language in a system for information servicing of collectivities, Serdica, vol. 4, fasc 2-3, 1978.**

5.П.Бърнев,К.Иванов, Входно-изходна система за информационно обслужване на колективи, IV Международный симпозиум по передачи данных и обработке информации, 26-31 мая 1977, Варна.

6.П.Бърнев, Вл. Занев, Ат. Раденски. Бази от данни. Система за информационно обслужване на колективи. VIII Пролетна конференция на СМБ, Сл. бряг, април 1979.

7.П.Бърнев,Ат. Раденски, Вл. Занев, Структуры данных в архиве системы информационного обслуживания коллективов, Сборник докладов конференции по приложении вычислительной техники и баз данных в научных исследованиях, Варна, май 1978.

8.Вл. Занев, Концепция и модель расспределенной системы информационного обслуживания коллективов, Сборник докладов конференции по ВЦКП, Благоевград, 1979.

9.П.Барнев, Кр. Марков, Проблемы управления в системах информационного обслуживания коллективов, Сборник докладов конференции по ВЦКП, Благоевград,1979.

MTA Számitástechnikai és Automatizálási Kutató Intézete, Tanulmányok 133/1982 Proc. of RG–11, KNVVT

Problems in modelling of data base performance

András Benczur

Data base systems have some special properties which make the modelling of their performance more difficult than that of computer systems. First we analyze these properties as differences from the models used in the theory of operating systems and from the mathematical model developed for information transmission. Then a new model - something like a conceptual data model - is proposed to describe the background of a data base; and, on the basis of this model, the main problems of data base performance are stated.

1. Special properties of data base systems

As it is shown in Fig.1, a Data Base System /DBS/ is inside a Computer System /CS/ which serves the information needs of an Information System /IS/.
A given realization of this information service uses certain parts of different CS resources. The theory of operating systems deals with the problem of the efficient use of these resources as a problem inside the CS. How we can decrease the amount of the used CS resources by better organization - this is the problem of CS performance. But the performance measures used in the evaluation of CS performance give no answer to the question how efficient our solution, i.e. our DBS, is from the aspect of the IS. It may happen that our solution requires a lot of data preparation work from the IS, e g. high redundancy in input, as it is typical in the case of independent batch subsystems using own files. This means that the IS must invest information work into the DBS, and I name this information investment. Having the "most efficient" solution of our DBS problem from the aspect of the CS, the decrease in information investment will result in a new DBS using more recources than the former one. This situation is illustrated in Figure 1.

So, the performance of a DBS is not only the problem of the efficient use of CS resources but that of an efficient information service as well. We cannot investigate this problem without a measure of information. This measure has to be defined on the information /or data/ model of the real world represented by the DBS. Data models developed in the last two decades are not capable of measuring information, they are oriented either to data manipulation purposes /e.g. file-systems, relational, hierarchical, and network data models/ or to knowledge representation /e.g. the proposals for conceptual data model/.

It is quite obvious to ask whether the measures and techniques developed on the basis of Shannon's entropy concept can be used or not. We have two problems when we try to describe the functioning of a DBS on the analogy of the information transmission model. The first one is that we have no background probability space where the information comes from. Our background space is not a mathematically well-defined one, the source information comes from a real "living" organization, so the background space is a continually developing, open system. The second problem is that the coding in the information theory is just the opposite of the coding in data bases. The classical coding theorems are concerned with coding and decoding signals /or finite series of signals/ coming in a sequence according to some probabilistic rule,/e.g. independent series, Markovian series etc./.

In a data base there is a very large code, the stored data. We have to change this code according to relatively small pieces of information /coding input data/ and we have to decode some part of this code according to planned or ad-hoc information request /coding output data/. This situation is shown in Figure 2. On this simple sketch, the source information is added to the stored data through an input coding system, and the answers to information requests are derived from the stored data by an output coding system. But we cannot say a DBS works like a data transmission channel with very large but finite

memory, because the output code is not the simple transmission of the source information, it is stimulated rather by the questions than by the source.

In order to clarify the role of the DBS further, we have to explain the role of the IS from a new point of view. As it is shown in Figure 3, an IS has two interfaces to the real world of the organization it is serving.One of them collects and makes managable facts about the objects constituting the organization. The set of these facts is the knowledge of the organization about itself. I call the corresponding interface the Information Elaboration System of the IS.The second interface understands the requests for information, and makes understandable the requested information for the requester, that is, it gives back an answer. I call this interface the Question and Answer Formulation System. The part of the organization where the requests arise I simply call the Information Using System. It covers, among other things, the decision system, or the operational systems of the organization.

The possible set of requests is denoted by Questions. Each question is connected to some part of the Knowledge, and the IS can determine the answer to the question from this information. The flow of information is the following:

the Q-A Formulation interface receives and understands the request, and through the IS, the question is connected to a part of the Knowledge. This part is not only accessible through the Information Elaboration Interface, but the answer can be determined from it as well. The answer is made understandable for the receiver by the Q-A Formulation interface. Naturally, some actions of this flow can be done in advance, and this is always true when the IS uses a CS.
One possible model for the information flow through the CS is also given in Figure 3.

2. Proposal for the background space of the IS: the Knowledge Space

The basic problem we have is to find a model which can satisfactorily play a similar role as the probability space does in Shannon's information theory. Since the probability space gives a total model of the information source, our model has to be a total model. Totality means that the model has to reflect the continually developing and growing self-knowledge of the organization. Before building up the model, that is the K-Space, we have to recall Fig.3. We only deal with items of knowledge which are accessible by the IS, that is, which are identified by means of formulizing a question. So a question is nothing else than the identification of a piece of knowledge, /or in other words, it is an entry point into the K-Space/, and the answer is the information contained in this information relevant for the receiver. While the K-Space contains total information, an answer is always limited: it gives only the relevant information. The three components mentioned below: a question, an answer and a piece of knowledge /or information/ make up a unit in the K-Space.

From another aspect, we can also say that a piece of knowledge corresponds to certain facts about an identifiable object in the real world. /This object can also be an abstract one, or a possible one and so on/. A question identifies both an object and the relevant facts about it. But the IS does not know the objects, it knows only the items of knowledge, and there is a one-to-one mapping from the set of objects to the set of accessible items of knowledge. So the structure of objects is reflected by the structure of accessible parts of knowledge. This structure contains not only natural but artifical aspects as well. The IS can identify the elements of this structure, that is,the parts of knowledge corresponding to the objects identified by the questions. From these parts of knowledge, the IS gives the answers like a filter. Figure 4 gives an illustration of this.

Let us turn to the definition of the K-Space.

Def.1. A threefold, /Q,A,I/ is called a K-element, where Q is a question, A is an answer and I is some information. The answer A is derived from I according to the question Q, so the pair Q and I completely determines A.

Def.2. A K-State $\mathcal{K}$ is given by the sets $\Theta = \{Q_i\}_{i=1}^{n}$, $\mathcal{A} = \{A_i\}_{i=1}^{n}$ and $Y = \{I_i\}_{i=1}^{n}$ as the set of K-elements $\mathcal{K} = \{(Q_i, A_i, I_i)\}_{i=1}^{n}$. The set Θ is the set of different questions, that is

$$Q_i \neq Q_j \quad \text{if } i \neq j .$$

$\mathcal{A}$ is the set of answers, and Y is the system of K-parts, which are the identified parts of the knowledge. For the sake of simplicity, we suppose, that both Q_i and A_i are contained in I_i, and A_i, being the actual relevant information, is accessible by Q_i. A K-part may contain another K-part, but the IS does only know it if it is embedded as a K-element. /See operation 2./

Def.3. A sequence of K-States is a K-Space, if each state is generated from the previous one by one of the two operations given below. This means a K-Space is a monotonous increasing sequence of K-States.

Operation 1. The addition of new information: the growing operation /G-operation/.

Let I_{new} be a piece of information which is going to belong to or form a K-part identified by Q. This means, some new information is added to the K-state through the entry point Q. The result is a new K-part with a new answer and its K-element contains the previous K-element.

We use the following notation

/1/ $$(Q, A, I) + (Q, I_{new}) = (Q, A_{new}, I + I_{new})$$

in the case when the K-element /Q,A,I/ is growing.

Remark 1 The notation $I + I_{new}$ means that the new K-part contains both the old and the new information, but they are not accessible separately.

Operation 2 The embedding of a K-element into a K-part: the embedding operation /E-operation/. When a new piece of information means that a part of knowledge is a subpart of another one and both of them are identified by a question, than the former one is embedded into the latter.

We use the following notation similar to /1/

/2/ $$(Q, A, I) \oplus (Q, [Q_1, I_e]) = (Q, A_{new}, I \oplus [Q_1, I_e]) = (Q, A_{new}, I + I_e \oplus Q_1)$$

Where Q_1 is the identifier of the embedded K-part and I_e is the embedding information that shows the role of the K-part identified by Q and determines the way of giving the new answer.

We use the notation $(Q_1, A_1, I_1) \in I$, or simply $Q_1 \in I$ if Q_1 is embedded into (Q, A, I).

The structure of the K-space is developed by E-operations. The following rules show the main features of the embedding:

Rule 1 Let (Q_1, A_1, I_1) be embedded into (Q, A, I), that is $Q_1 \in I$. If some new information, I_{new}, is added to Q_1, it affects the K-element (Q, A, I), too. In notational form supposing

/3/ $$I = I' \oplus Q_1 = I' \oplus (Q_1, A_1, I_1)$$

we have

/4/ $$(Q, A, I' \oplus (Q_1, A_1, I_1)) + (Q_1, I_{new}) = (Q, A_{new}, I' \oplus (Q_1, A_{1new}, I_1 + I_{new}))$$

This means that an embedded K-element remains embedded in its actual state automatically, and its changes consequently change its host K-element.

Rule 2 It is similar to Rule 1, but changing the G -operator in/ 4 / into an E-operator.

Remark 3 Since there are no restrictions given for E-operations, a K-state may have a very complicated structure full of deep recursion. Rule 1 and Rule 2 do not determine the sequence of changing the host K-elements, so it is supposed that the corresponding changes are made simultaneously. From this reasoning it follows that the G - and E-operations rather affect the whole K-state than one of its K-elements.

Remark 4 I only want to mention here I have tried out how the K-space can reflect different data models.

3. The measure defined on the K-space

There are two aspects that mainly determine the construction of our measure. The first one is that questions, answers and information parts are understandable for the organization and each of them has an information quantity, which is necessary for understanding. This means, these quantities are different from the code lengths of an "optimal" coding.

The second aspect is that during the development of the K-space, the quantity of each K-part gets larger by the new facts added to it by a G- or E-operator according to Rule 1 and Rule 2.

Def.3 The K-measure $\varkappa$ is defined on each component of the K-space, that is on each question, answer, K-part and K-element. The measure $\varkappa$ satisfies the following conditions:

Cond.1 Within a K-state given by the K-elements

$$(Q_i, A_i, I_i) \quad , \quad i = 1, 2, \ldots, n ,$$

$$\sum_{i=1}^{n} 2^{-\varkappa(Q_i)} < 1 .$$

The meaning of this condition is, that questions are different,

and we also need the possibility to add new questions to a state.

Cond.2 Within a K-element /Q,A,I/ the K-part I contains both Q and A, so

$$\mathcal{x}(I) \geq \mathcal{x}(Q) \quad \text{and}$$

$$\mathcal{x}(I) \geq \mathcal{x}(A).$$

Cond.3 The measure of a piece of information to be added to the K-state expresses its independent content, so the measure of each K-element that contains the K-element to which the new information is added by a G-operation is increased by the measure of the new information. According to /1/ and /3/ this means

$$\mathcal{x}\{(Q,A,I) + (Q,I_{new})\} = \mathcal{x}(I) + \mathcal{x}(I_{new})$$

and

if $Q \subset I_1$

$$\mathcal{x}\{(Q_1,A_1,I_1) + (Q,I_{new})\} = \mathcal{x}(I_1) + \mathcal{x}(I_{new}).$$

Cond. 4 The embedding of a K-element into another one increases the measure of the host only by the measure of the part of the graft that is not known by the host. So, according to /2/ if $Q_1 \not\subset Q_2$ and $Q_2 \not\subset Q_1$ hold,

$$\mathcal{x}\{(Q_1,A_1,I_1) \oplus (Q_1,[Q_2,I_e])\} = \mathcal{x}(I_1) + \mathcal{x}(I_e) + \mathcal{x}(Q_2) + \mathcal{x}(I_2).$$

And in the case when $Q_1 \subset Q$ and $Q_2 \subseteq Q$

$$\mathcal{x}\{(Q,A,I) \oplus (Q_1,[I_e,Q_2])\} = \mathcal{x}(I) + \mathcal{x}(I_e) + \mathcal{x}(Q_2).$$

After building up our measure, we can measure the work of the IS necessary to execute G- or E-operations. We can determine this work by adding all the values that express the growths of all K-elements affected by the operation.
For example, if Q is embedded in $Q_1, \ldots, Q_k$ and we add (Q, I_{new}) to the K-state, the volume of the execution work is equal to

$$(k+1)\,\varkappa(I_{new})$$

The derivation work of an answer in the K-element (Q,A,I) is the difference $\varkappa(I) - \varkappa(A)$

Note The investigation of the properties of the K-measure has not been finished yet. It may have some weaknesses and the complexities of the structure developed by E-operations may need some measure, too.

4. Formulation of the DBS effectivity

In this last paragraph, we describe the problem of the DBS effectivity in terms of the K-space.
First we characterize the DBS as it reflects some subparts of the K-space.
The most important subpart of the K-space, which determines the DBS, consists of the set of possible questions coming from the Information Using System; $\Theta_0 \subset \Theta$ ./This type of short notation means, that, when the K-space grows, both Θ_0 and Θ grow./ Θ_0 determines the set of answers $\mathcal{A}_0$ that corresponds to the output of the DBS.

Since the answers in $\mathcal{A}_0$ are changed in consequence of G and E-operations, it is not suitable that the DBS should contain only answers. So we can say that the DBS contains mappings of special K-parts that are sufficient for deriving the answers.

The set of these K-parts is given by the set of questions $\Theta_D \subset \Theta$. The stored data do not cover the total information contained in the K-elements $/Q_i, A_i, I_i /$, $Q_i \in \Theta_D$ Instead of I_i the DB contains some data D_i mapped from I_i, so that it is sufficient to derive all the answers belonging to Q_j, $Q_j \in \Theta_o$. We can say that the system $/Q_i, D_i/$, $Q_i \in \Theta_D$ gives the logical model of the stored data.

The third important part of the DBS is the system of special K-elements that are embedded in K-elements belonging to Θ_D. The special property of these elements is that new information are added to them by G-operations, and they are most frequently embedded by E-operations. The system of these elements is given by the set of questions Θ_I, and the new pieces of information added to them are the input of the DBS.

Now we can express the efficiency of the DBS in the following way:
The information source is given by a stochastic point process, the events of which mean the growth of the information space restricted to Θ_I. We can associate with the events the measure of the information growth in Θ_D and Θ_o caused by the corresponding G- or E-operation. The expected average value of this information growth can be treated as the source capacity.

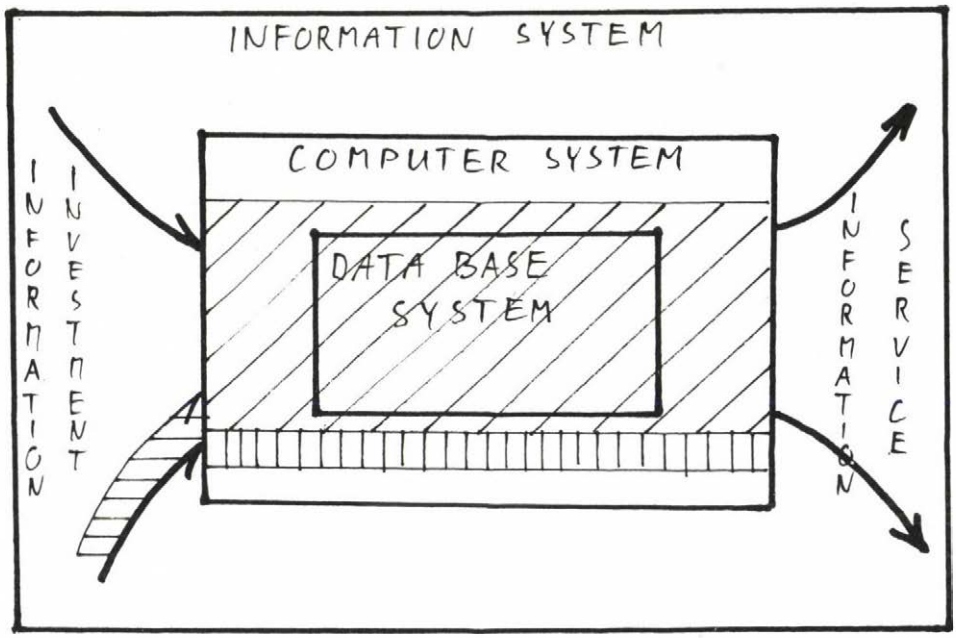
The stored information /the content of the DB/ can be measured as a part of the information space. An upper bound can be given by adding the values $\varkappa(I_i)$ when $Q_i \in \Theta_D$. This upper bound corresponds to the solution of highest redundancy. This upper bound karacterizes the necessary storage capacity.

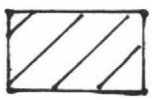
The third capacity corresponds to the stream of questions coming from the Information Using System.

This capacity can be given similarly to the input capacity: a stochastic point process describes the occurrance of questions,

and the value of $\varkappa(Q_i) + \varkappa(A_i)$ is associated to the events of the process. Maybe it is useful to separate the measure of questions and that of the answers, because question coding and answer derivation may use different processes.

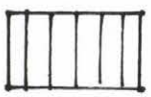
These capacities require different computer system resources, /extended sometimes by human power of different computer - specialists/. Efficiency is: to use fewer and cheaper resources to assure the same capacities. The most interesting point in this problem is the complicated conflicts among the capacity demands in using computer-system resources. For example, the more recources are used to update the data base code by the input, the fewer recources are needed to meet the output demands.





used resources of CS

decrease in Information Investment



extra resources corresponding to decreased Information Investment

Figure 1

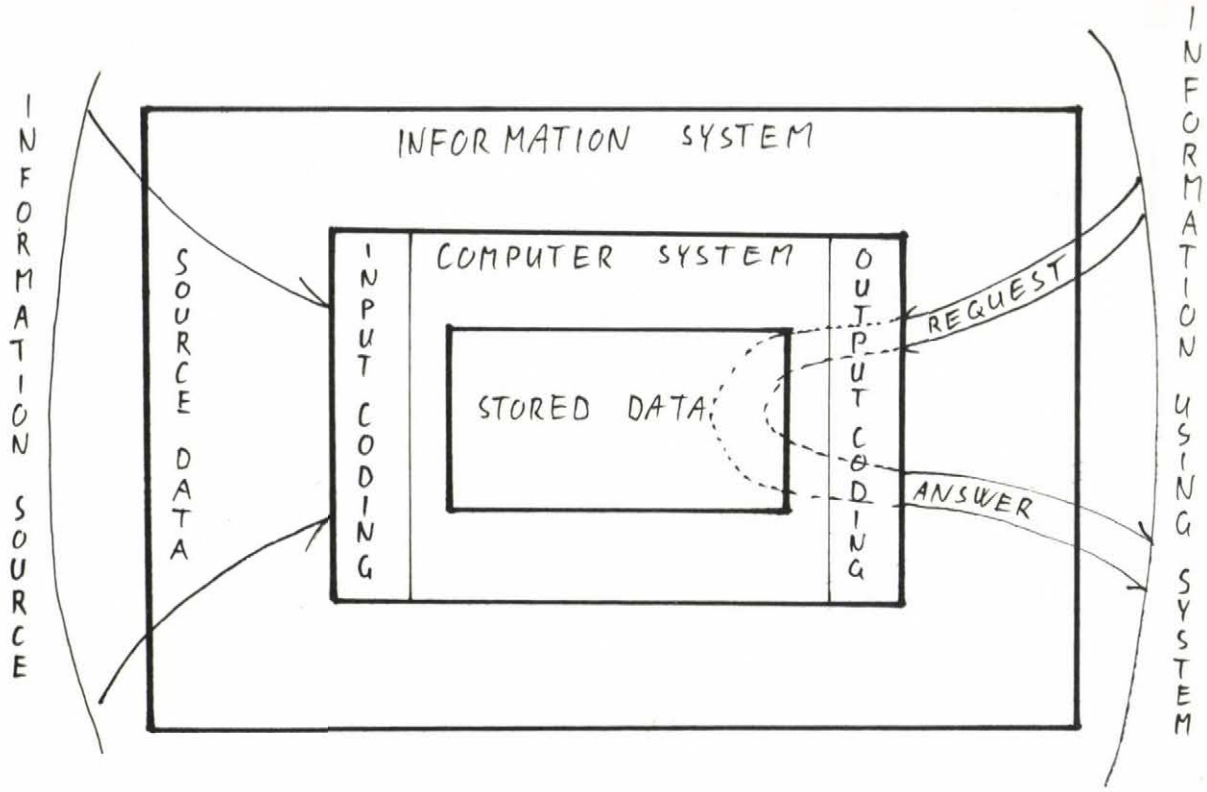

INFORMATION SOURCE
INFORMATION SYSTEM
SOURCE DATA
INPUT CODING
COMPUTER SYSTEM
STORED DATA
OUTPUT CODING
REQUEST
ANSWER
INFORMATION USING SYSTEM

Figure 2

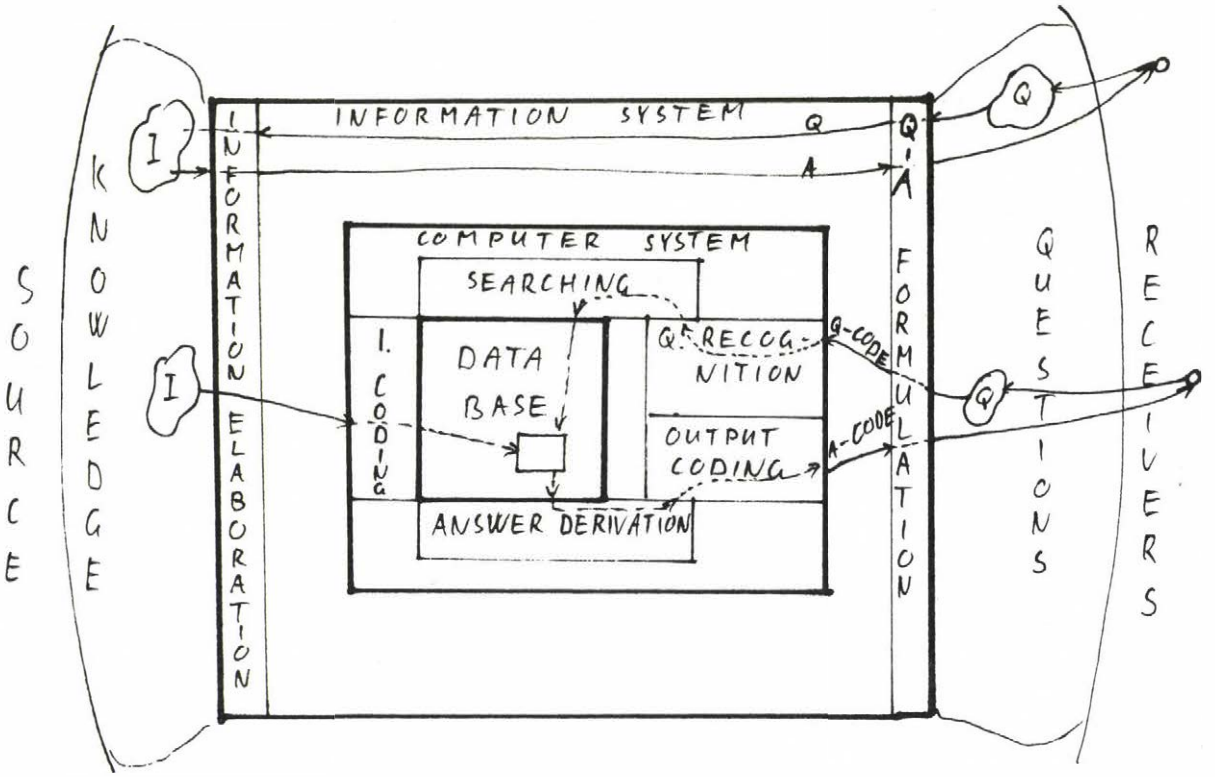

SOURCE
KNOWLEDGE
I
INFORMATION ELABORATION
INFORMATION SYSTEM
Q
A
Q-A FORMULATION
COMPUTER SYSTEM
SEARCHING
I. CODING
DATA BASE
Q. RECOGNITION
OUTPUT CODING
ANSWER DERIVATION
Q-CODE
A-CODE
Q
QUESTIONS
RECEIVERS

Figure 3

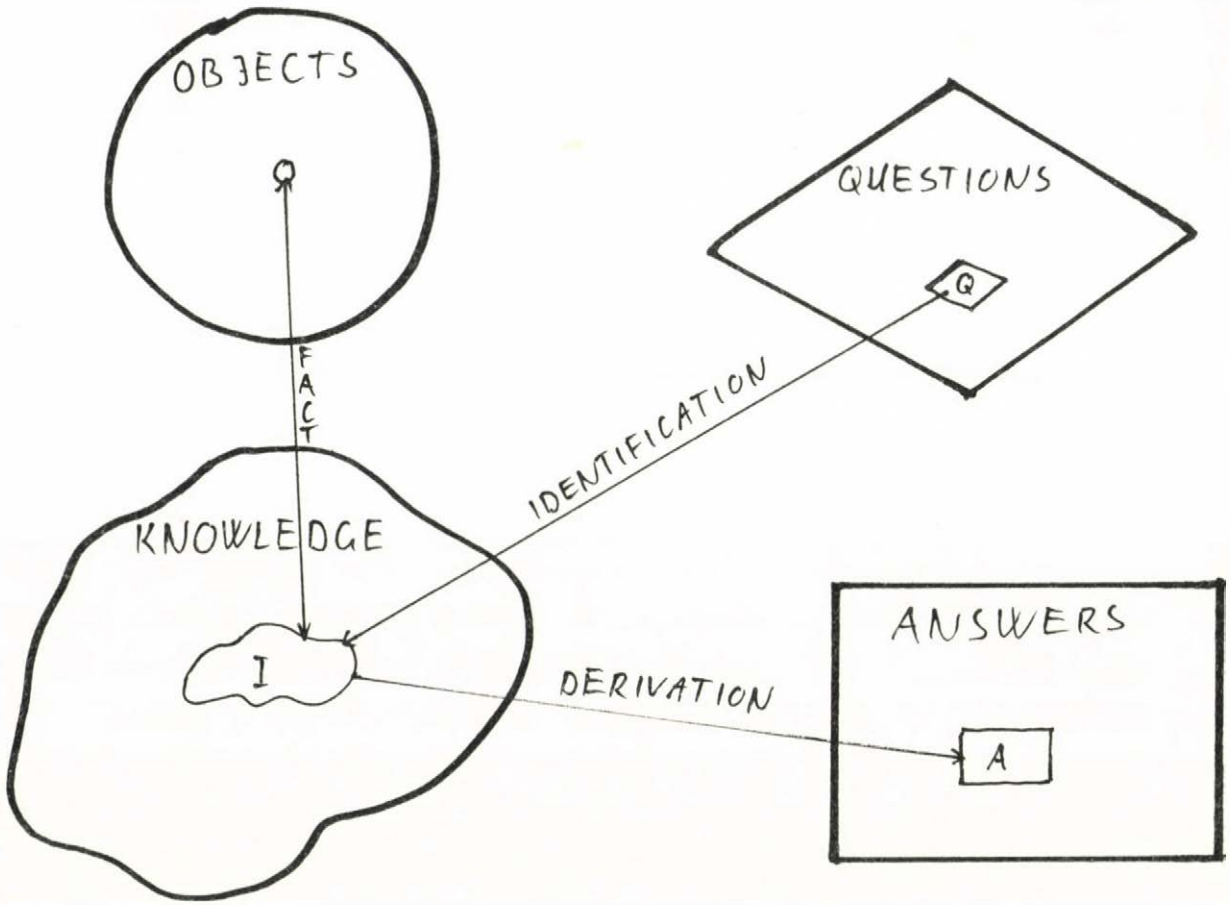

OBJECTS
QUESTIONS
Q
FACT
IDENTIFICATION
KNOWLEDGE
I
DERIVATION
ANSWERS
A

Figure 4

MTA Számitástechnikai és Automatizálási Kutató Intézete, Tanulmányok 133/1982 Proc. of RG–11, KNVVT

DBS/R - A SYSTEM OF PRACTICE

JÜRGEN BITTNER

VEB ROBOTRON ZENTRUM
FÜR FORSCHUNG UND TECHNIK
8012 DRESDEN
PSF 330

1. Introduction

In the GDR since 1979 the efficient Data Base Management System/Robotron (DBS/R) has been available. It has been developed in VEB Robotron Zentrum für Forschung und Technik (Centre for Research and Technology) and tested together with enterprises and institutions of different branches of economy. Until April 1981, 136 buyers have purchased this system; some of these sales will include the manifold reuse in companies of whole industrial branches. Averagely at present two to three projects per user are prepared or used. Both databases for support of projects of the single traditional fields and also highly integrated systems up to management information systems are represented in the large number of applications. In addition to the advantages of database organization, these applications utilize the specific advantages of DBS/R for a rational projecting which permits a short-term achievement of the objects in view.

2. Short Characteristic of DBS/R

Data and storing structure

DBS/R permits both on the level of record types and also of

record instances the representation of optionally extensive network structures. The main components of the structure - master and chain data - have a very high flexibility and can be adapted well to the specific logic data structure of the user. DBS/R provides this basic structure on an essentially higher convenience level than it is know, for example, by TOTAL of Cincom Systems or DBOMP of IBM. For implementation, DBS/R uses address reference, address chains, primary index, and secondary index. As to the efficiency these techniques may be used very differentially. DBS/R supports traditional file organizations, too, to make possible a uniform methodology for different data sets.

Data Description

A special subsystem, the Data Description Catalog (DATKAT), serves for creating and updating the data descriptions. It permits the development of a concrete data structure by steps and provides simultaneously the possibility for documentation. In addition to the description of the database (files, records, segments, fields, relations between records, integrity conditions etc.), this subsystem performs the definition of all inputs for the database as well as their allocation to the elements of the database.

Data manipulation

DBS/R has a very efficient data manipulation language. It can be used in connection with the basic languages Assembler, PL/1, COBOL (host language) and also independently on other languages (self-contained), and is easy to learn. This language comprises both statements with elementary services and also very complex statements. The self-contained application of the DBS/R language permits in addition to the fast formulation of inquiries and reports the design of routine programs for frequent use, too.

The system has statements for the access to single records and record sets of the database, for input and output of sequential data sets, the layout of lists, for comparing, for arithmetic operations as well as for the representation of favourable program structures.
The statements for creating and updating the database offer a high performance. One single statement can control here the processing of an input file which releases updatings in the total database including extensive defined checks.

Additional functions

Essential additional functions are

- data privacy
- securing the physical integrity
- restructuring

By means of data privacy the user organizes the protection against unauthorized use of the database system. This is performed variously according to data units of different level, copies of data units as well as to functions and function groups. The components for securing the physical integrity preserve the user from the loss of the database or of intermediate results of programs in consequence of machine, program, or operating errors. Furthermore, they make possible the recovery of older database states.

Restructuring serves for changing the structure of the database, which may become necessary in the interest of an increase of effeciency or of modifications of the real problem definition. For this purpose, a special status of data description is provided for transferring the database to the new structure.

Modes of operation

DBS/R can be used both in batch processing and also in real-

time mode with teleprocessing.

Purposeful procedures support singly or together the generation of programs, their storing and execution for the batch processing. For real-time operation the Exekutivsystem is to be generated. This system controls processing of transactions, commands, and messages for up to 250 terminals. Figure 1 shows the essential components of the Data Base Management System.

General prerequisites

The system operates on all ESER systems using the OS/ES Operating Systems from VEB Kombinat Robotron, e. g., EC 1022, EC 1035, EC 1040, EC 1055. Furthermore, DBS/R is able to work on systems as IBM/360 and IBM/370 with the appropriate operating systems. The Exekutivsystem requires generation of the TCAM access method.

3. Application of DBS/R

3.1. Fields of application

In the GDR, DBS/R is used in almost all branches of economy and institutions of public life. Nearly all industrial branches work with data bases on the basis of DBS/R. The most representatives come from the enterprises and companies of the following ministries (table 1):

Table 1: Application of DBS/R in the industrial branches of GDR

	number of companies
electrical engineering/electronics	41
heavy machine and plant construction	14
chemical industry	11
general mechanical engineering, agricultural machine and vehicle construction	9

	number of companies
machine tool and processing machines construction	27
building industry	4
ore mining/metallurgy	15
light industry	8

Furthermore, branches such as agriculture, forestry, food-stuffs economy, posts and telegraphs, traffic, public health, higher education, geology, culture, local authorities, etc. are represented. Nearly all users of the branches mentioned have own computing centres. They prepare the use of DBS/R independently by means of the normative support services of VEB Robotron ZFT.

A large number of smaller enterprises use DBS/R by means of the regional service computer centres of the GDR who have acquired the system for extensive reuse. A further increase of DBS/R application in the GDR is to be taken into account. Also the export to the countries Hungary, Iraq, India, China, Cuba, Corea, Angola has begun. Realized projects exist in Hungary, China, and India.

The most extensive field of application at present is the technical preparation of production with about 40 to 45 % of the projects. The share of other applications such as material economy, manpower, sales, production planning, balancing, investment planning as well as of applications in non-producing spheres increases very much. The share of DBS/R applications in technical preparation of production will not exceed, however, 25 % of the total number of applications also in the next years. First management information systems based on DBS/R are already in use, too.

3.2. Characteristic examples

3.2.1. DBS/R in technical preparation of production

A large number of application have a database the core of

which contains the constructive and technological master data of products and subassemblies to be produced (figure 2). It offers excellent prerequisites for their updating. With a high quality, from this database bills of materials, using lists, routing sheets etc. can be demanded both in batch and also in real-time mode.

This part of database has been supplemented at many users by files for item designations, operation texts, routing header data, variant structures. In some cases, constructive and technological bills of materials are stored in particular files. Frequently these data have been integrated with further complexes which are still organized in different enterprises as isolated databases. Such a complex is material economy. It comprises material stock keeping and checking, material statistics, orders and supplier master data. For purposes of planning and control of production, the database contains an order file which corresponds with a structure specification in many cases. Thus, a rational producing of the production order documents will be possible.

The performance of these databases at several users will be extended by the following data complexes:

- capacity data for workplaces or workplace groups
- integration with data of sale consisting of customer master data, contract data, deliveries, invoices
- integration with costs data such as material costs and wage cost as well as payroll accounting data

The DBS/R applications in these fields have resulted in very good effects. An examination of benefits achieved by means of DBS/R in technical preparation of the enterprises in the sector of the Ministry of Electrical Engineering and Electronics showed the achievement of an essential reduction of the time required for providing the constructive and technological documents. Averagely 12,000 labour hours of

technologists were saved annually per enterprise.

The high convenience of the system, especially of the efficient DBS/R language, has made possible furthermore an 80 % saving of designing capacity. Many problems such as amendment service and bill of materials explosion have been solved exclusively by DBS/R language statements. Especially for ascertaining the total demand for one or several products, services of DBS/R are used permitting the development of very fast programs. Operative inquiries which cannot be foreseen will be programmed and executed in a few minutes.

An example for a simple program for creating a bill of materials is shown in figure 3.

3.2.2. DBS/R as a basis for a personnel database

Large-size industrial enterprises of the GDR use personnel databases consisting of the following four complexes:

- personnel data
- planned job data
- structure data
- designation data

Their logical structure is shown in figure 4.

Databases with this or a similar structure have proved good already in many cases. The event-referred amendment service (salary change, change of job, qualification etc.) utilizes particularly the variable input formats of DBS/R by which a great number of different acquisition vouchers are represented. The great number of logical checks formulated in the data description is very advantageous in the interest of correctness of data. Periodical routine tasks such as wages and salaries accounting, services for the working people, e. g., rent remittance, insurances etc., are performed with a high efficiency. Special inquiries and

evaluations support very efficiently the daily work of the clerks in the personnel department. Payroll accounting and similar complicated routine programm have been programmed to great extent in PL/1 and embeddeo DBS/R statements. Also controlling DBS/R programs with calls for PL/1 or Assembler modules are used. A great number of search inquiries and total evaluations of the database have been performed as pure DBS/R programs. Especially it has been succeeded to introduce the use of DBS/R language statements or of a suitable subset, respectively, for formulating ad-hoc evaluations and inquiries by clerks of the personnel department.

3.2.3. DBS/R application in the management information system

Management information systems on the base of DBS/R have been developed in the GDR for enterprises, combines, and ministries and have been already used in some cases. The database supports there problems of planning, balancing, accounting, reporting as well as the check of time scheduling and regulations. Figure 5 shows a section from such a database. These applications have special requirements for the real-time behaviour of the total system. The DBS/R Exekutivsystem has stood the test there. For securing a fast access (1...5 sec) to especially important information the database structure contains special image memories.

A high application convenience for final users is offered by parameter-controlled evaluation program systems developed by means of the DBS/R language. Operation is supported by additional communication components. They make easy the contact of the final users to DBS/R. It is typical for such databases, that a great part of data is supplied by other - partially non-uniform - EDP projects. Here, DBS/R offers very good prerequisites for transferring the most different input data.

Solutions for the entire problem circle of management information systems are offered by Kombinat Robotron in the scope of the MIS/R system.

4. Conclusions

The widespread use of DBS/R in the GDR is also a result of the service offered by Kombinat Robotron to their users. The training program of the Training Centre Leipzig has contained DBS/R courses for the basic and the Exekutivsystem for several years. Furthermore, the designer performs user-specific training and will be always at the users's disposal for consultations. First good results have been achieved also in the PR Hungary with DBS/R. It is to be emphasized that there in a very short time a database has been installed at CSEPELAUTO which is the result of an intensive work of the CSEPELAUTO team and a good cooperation with the DBS/R designer.

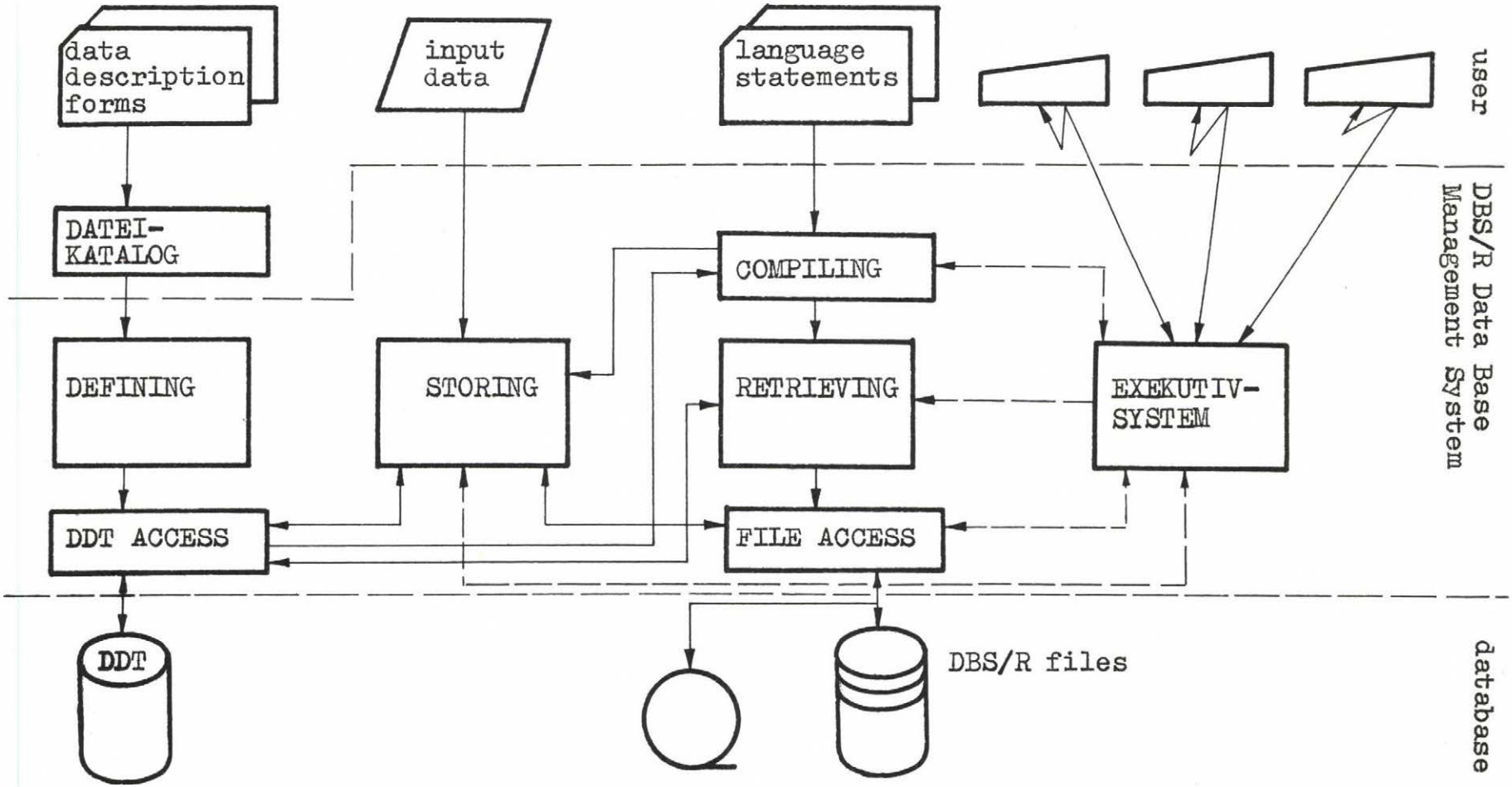
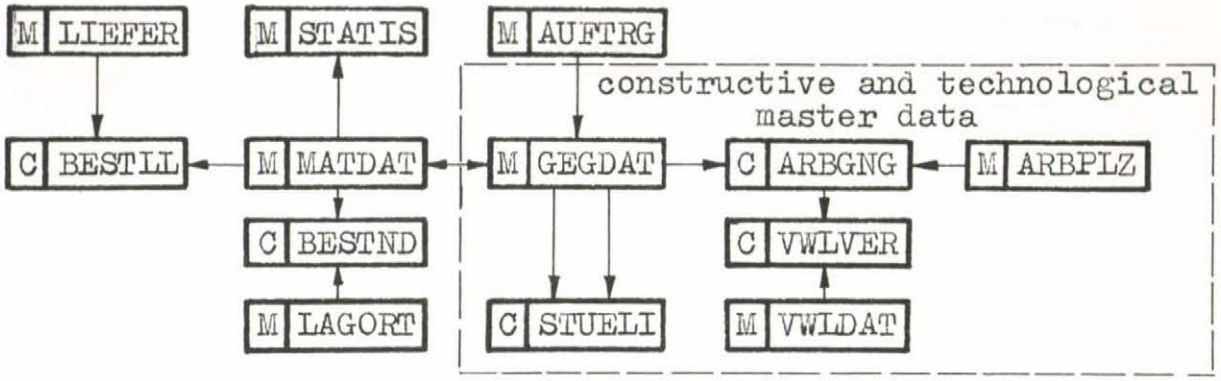


Figure 1: Components and modes of operation of DBS/R



GEGDAT Master file for items (products, subassemblies, parts)

ARBPLZ Master file for workplaces

VWLDAT Master file for fixtures, tools, gauges

ARBGNG Chain file for operations chained per item (operation sequence) and per workplace (workplace using list)

VWLVER Chain file for fixtures, tools, gauges, chained per operation (demand for fixtures, tools, gauges) and per FTG (FTG use)

STUELI Chain file of the item structure (bill of materials items) chained per product (subassembly bill of materials) and per component (parts using list)

MATDAT Master file for material master data coupled with GEGDAT

LAGORT Master file for storage places

STATIS Master file for material statistics coupled with MATDAT

BESTND Chain file for material stocks chained per article (stocks of one article in different storage places) and per storage places (stocks of different articles of one storage place)

LIEFER Master file suppliers

BESTLL Chain file for orders chained per supplier and per article

AUFTRG Master file for orders coupled with GEGDAT

Figure 2: Structure of a database for technical preparation of production and material economy

a) DBS/R program

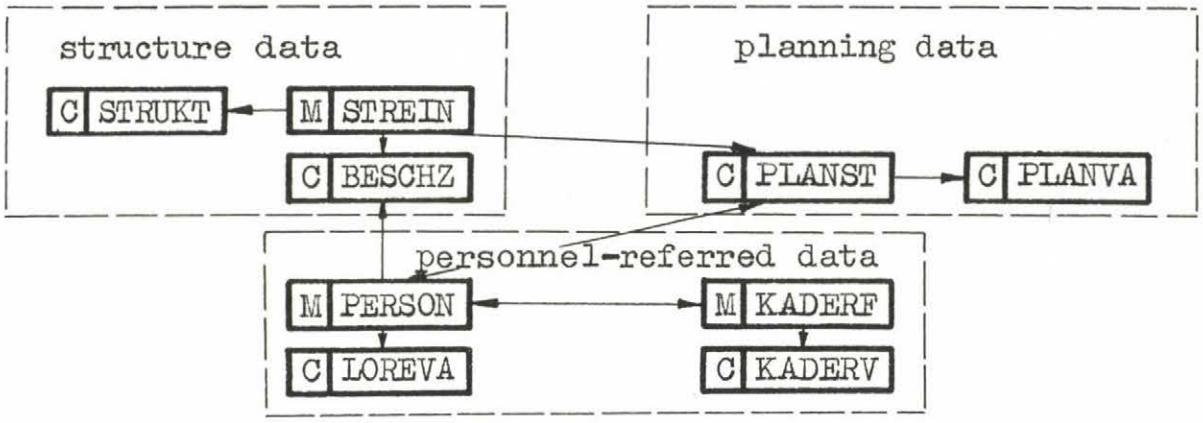
```
JCB: KOCH10      STEP: DBS      PROGRAM: DBS#R                10.06.81   10.37   PAGE   1
                                          SYSPROT1 LOG

===   DBS/R 3.1   ===
                    RETRIEVE EXECUTE;                                          1
                    DECLARE RLEVEL 8;                                          2
                    DECLARE IAREA RECORD 13;                                   3
                    DECLARE KEY 13;                                            4
                    INPUT TO IAREA;                                            5
                    READ FROM MMPART BY A KEY;                                 6
                    MOVE FROM MMPART PARTNO    'RPARTNO'                       7
                         FROM PARTNAME    'RPARTNAM';                          8
                    OUTPUT;                                                    9
                    HEADING 'IMPIODING LIST' RPARTNO RPARTNAM;                 10
                    FEED;                                                      11
                    HEADING 'RES.LEV. PART NUMBER   NAME OF PART    #          12
                             #          UM'   5;                               13
                    FEED;                                                      14
                    HEADING;                                                   15
                    STRUCTURE VIA MMPART IN MCSTRU WITH STRUCLEVEL             16
                              EXTERNAL TO RLEVEL ON STRUCEX2=OUTPUT;           17
          OUTPUT    OUTPUT;                                                    18
                    LINE RLEVEL 5 MMPART PARTNO 15 PARTNAME 30 UM 65;          19
```

Figure 3: Creation of a bill of materials

b) structure bill of materials

IMPLODING LIST 11 AMPLIFIER

RES.LEV.	PART NUMBER	NAME OF PART	UM
.1	12		ST
.1	14	TRANSISTOR	ST
.1	15	circuit board, fitted	ST
..2	16	WIRE BUNDLE	ST
...3	17		M
...3	18		ST
..2	12		ST
..2	13	TRANSISTOR, BALANCED	ST
...3	14	TRANSISTOR	ST



STREIN Master file for basic units of all levels of the company

STRUKT Chain file for representation of the hierarchical relations between the basic units

BESCHZ Chain file for membership of persons to basic units

PERSON Master file for general person-referred master data

LOREVA Chain file for variable data for payroll accounting and payroll accounting result data chained per person

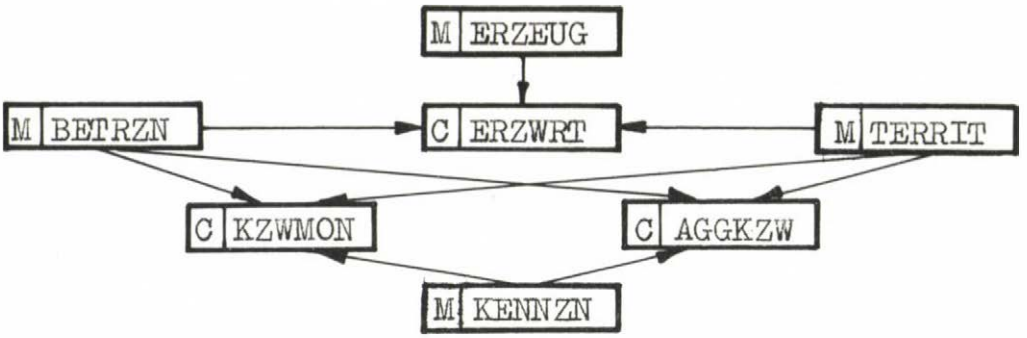
KADERF Master file for staff-specific master data coupled with PERSON

KADERV Chain file for variable staff data

PLANST Chain file for planned jobs chained per basic unit

PLANVA Chain file for variable data per planned job

Figure 4: Structure of a database for personnel system



ERZEUG	Master file for master data of products
BETRZN	Master file for enterprises and companies
TERRIT	Master file for territorial units
ERZWRT	Chain file for monthly quantities of products chained per enterprise, product and territory
KENNZN	Master file for types of characteristics
KZWMON	Chain file for monthly characteristic values of enterprises, chained per enterprise, type of characteristic or territory
AGGKZW	Chain file for characteristic values aggregated per branches of industry, territories, periods

Figure 5: Section of a database structure of a management information system for a ministry

MTA Számitástechnikai és Automatizálási Kutató Intézete, Tanulmányok 133/1982 Proc. of RG-11, KNVVT

LOGICAL DEPENDENCIES IN RELATIONAL DATA BASE

J.Demetrovics - Gy.Gyepesi

§0. INTRODUCTION

According to E.F.Codd [6] a relation is a matrix without two identical rows. Rows correspond to data records and columns to the attributes that are to be stored of a data item. He also introduced [7] the concept of functional dependency: a set of columns depends on another if fixing the values in a row taken on the first determine those on the second.

Other concepts of his are the key (a set of attributes on which all depend) and the candidate key (a minimal key).

Candidate keys clearly do not contain each other [12].

The possible mathematical structure of functional dependencies was first investigated by W.W.Armstrong [1]. Among others he found that this structure is determined by the maximal dependencies (those which have maximal attribute subsets depending on minimal ones) and even by the dependent sides of the maximal dependencies. We also heavily use these "maximal dependent subsets of attributes" as technical tools.

Different kinds of functional dependency have also been introduced [3], [9], [13], [15] and axiomatized, usually in similar systems to Armstrong's [8]. [10] discusses an interesting connection between the decomposition of relational data bases and the boolean switching functions.

The harder problems of the topic are usually of combinatorial nature (see [4], [5], [11], [16]).

In this paper in §1 we give the formal definition of the functional, dual, strong and weak dependencies and give new axioms for full f-d- and s-families.

In §2 develop the analogy and differences among the dependencies of different types and give an axiom for full w-families.

In §3 we deal with a question stated in [11].

Certain dependencies of a relational data base are known by its designer. We call these initial dependencies. In general initial dependencies imply new dependencies. W.W.Armstrong [2] has developed a method to find the dependencies implied by a given set of initial functional dependencies. He also gave a characterization of the sets of initial dependencies that imply all the dependencies of a given full f-family and are of minimal cardinality. This characterization has a logical nature; we give a combinatorial equivalent of it.

We use the following notational conventions:

Ω denotes the set of attributes, $P(\Omega)$ denotes his power set. If g is a function with X as its domain and $Z \subseteq X$ then $g\restriction_Z$ denotes the function which has domain Z and for any $z \in Z$ $g(z) = g\restriction_Z(z)$. $\subset$ means strict inclusion.

§1. OLD AND NEW AXIOMS

We start with the definitions of functional, dual, strong and weak dependencies based on [1] and [8].

Definition 1.1 Let A,B be subsets of Ω and let R be a relation over Ω. Then we say that B

(i) functionally;
(ii) dually;
(iii) strongly;
(iv) weakly

depends on A in R if

(i) $(\forall g,h\in R)\ (g\restriction_A = h\restriction_A \rightarrow g\restriction_B = h\restriction_B)$;

(ii) $(\forall g,h\in R)\ ((\exists a\in A)(g(a)=h(a))\rightarrow(\exists b\in B)(g(b)=h(b)))$;

(iii) $(\forall g,h\in R)\ ((\exists a\in A)(g(a)=h(a))\rightarrow g\restriction_B = h\restriction_B)$;

(iv) $(\forall g,h\in R)\ (g\restriction_A = h\restriction_A \rightarrow (\exists b\in B)(g(b)=h(b)))$.

holds respectively and denote these by

$A \overset{f}{\underset{R}{\rightarrow}} B$, $A \overset{d}{\underset{R}{\rightarrow}} B$, $A \overset{s}{\underset{R}{\rightarrow}} B$, $A \overset{w}{\underset{R}{\rightarrow}} B$ corresponding to the type of the denoted dependency.

The following example [8] illustrates the effect of the dual dependency.

Example: Let Ω = {author, title, hall, shelf}. Let we have a library with eighteen books, three halls and three shelves in every hall; one shelf holds two books. Let the relation R containing the datas of the library given by the following table:

author	title	hall	shelf
1	1	1	2
2	2	1	3
3	3	1	1
4	4	1	2
5	5	2	3
6	6	2	1
7	7	2	2
8	8	2	3
9	9	3	1
10	10	3	2
11	11	3	3
12	12	3	1
1	4	1	1
5	8	3	3
4	1	1	3
7	10	3	2
6	10	2	2
6	9	2	1

Thus {author, title} $\overset{d}{\underset{R}{\rightarrow}}$ {hall, shelf} holds, and for i=1,...12 the book by author i and entitled i is on the $(1+3\cdot\{\frac{i}{3}\})$ -th shelf of the $[\frac{i+3}{4}]$-th hall ($[x]$ denotes the whole part and $\{x\}$ the fraction part of x). The reader, knowing the author or the title of the

required book, may find it without examining the whole library: for example if $\underline{i}$ is the author of the book, then it is enough to look the $[\frac{i+3}{4}]$-th hall, and the $(1+3\cdot\{\frac{i}{3}\})$-th shelves of the other two halls.

In R {author, title} $\xrightarrow{f}_R$ {hall, shelf} holds too, but to store this functional dependency is equivalent to store the table of R; the {author, title} $\xrightarrow{d}_R$ {hall, shelf} dependency is more effective.

For proving the effectiveness of these dependencies we elaborated in the Automation Institute of the Hungarian Academy of Sciences a large-sized practical application of the relational data model.

We have planned an inventory-recording system for an agricultural corporation. The task of the system is to organize the component-traffic of about 350 agricultural estates. More exactly the task is: to record the inventory-stores, the orders of customers, to help the decisions making in this field and to help services.

First we used a traditional system concept for this purpose. Later this concept was transformed into the relational data model based on recent investigations. We saved about 40 percent of the memory capacity in this way. With using the results of Aho, Sagiv and Ullmann about relational expressions, we proved that the response time remained in the same order.

If R is a relation over Ω, and $Y\in\{F,D,S,W\}$ and $y\in\{f,d,s,w\}$ corresponds to Y then we use the notation

$$Y_R = \{(A,B): \quad A \xrightarrow{y}_R B\}.$$

We call <u>full y-families</u> the sets having this form.

In order to investigate the various dependencies the first step is the axiomatization of full y-families for $y\in\{f,d,s,w\}$. In [1] there is a system of axioms for full f-family and in [8] there are for full d- and s-families. For the sake of completeness we reproduce them here.

Let $Y \subseteq P(\Omega) \times P(\Omega)$. Then we say that Y satisfies the f-axioms, if for all $A,B,C,D \subseteq \Omega$

(F1) $(A,A)\in Y$:

(F2) $(A,B)\in Y$, $(B,C)\in Y \rightarrow (A,C)\in Y$;

(F3) $(A,B)\in Y$, $A \subsetneq C$, $D \subseteq B \rightarrow (C,D)\in Y$;

(F4) $(A,B)\in Y$, $(C,D)\in Y \rightarrow (A\cup C, B\cup D)\in Y$.

Y satisfies the ϑ-axioms if for all $A,B,C,D \subseteq \Omega$

(D1) $(A,A)\in Y$;

(D2) $(A,B)\in Y$, $(B,C)\in Y \rightarrow (A,C)\in Y$;

(D3) $(A,B)\in Y$, $C \subseteq A$, $B \subseteq D \rightarrow (C,D)\in Y$;

(D4) $(A,B)\in Y$, $(C,D)\in Y \rightarrow (A\cup C, B\cup D)\in Y$;

(D5) $(A,\emptyset)\in Y \rightarrow A = \emptyset$.

Y satisfies the γ-axioms if for all $A,B,C,D \subseteq \Omega$

(S1) $(\{a\},\{a\})\in Y$;

(S2) $(A,B)\in Y$, $(B,C)\in Y$, $B \neq \emptyset \rightarrow (A,C)\in Y$;

(S3) $(A,B)\in Y$, $C \subseteq A$, $D \subseteq B \rightarrow (C,D)\in Y$;

(S4) $(A,B)\in Y$, $(C,D)\in Y \rightarrow (A\cap C, B\cup D)\in Y$;

(S5) $(A,B)\in Y$, $(C,D)\in Y \rightarrow (A\cup C, B\cap D)\in Y$.

We need the following technical lemma.

Lemma 1.1. Let $F \subseteq P(\Omega) \times P(\Omega)$ be such that $(X,Y)\in F$ and $Y \neq \emptyset$ imply $X \neq \emptyset$. Then F satisfies the f-axioms iff $D = \{(A,B):(B,A)\in F\}$ satisfies the ϑ-axioms.

Proof: Trivial by the f-and ϑ-axioms. (D5) makes necessary the assumption that $(X,Y)\in F$ and $Y \neq \emptyset$ imply $X \neq \emptyset$. □

Remark: The assumption ($(X,Y)\in F$ and $Y \neq \emptyset$ imply $X \neq \emptyset$) in lemma 1.1 is not an important restriction:
if F satisfies the f-axioms let $F' = F \setminus \{(\emptyset,X):X \neq \emptyset\}$.
Then F' obviously satisfies the f-axioms and the critical assumption as well and we have: $X \neq \emptyset$ implies that $(X,Y)\in F \leftrightarrow (X,Y)\in F'$.

In the following we give new axioms instead of the f- ϑ- and γ-axioms and give an axiom that characterizes the weak full w-families which is such a full w-family that whenever (X,Y) is an element of the family then X is not void.

F-axiom: Let $F \subseteq P(\Omega) \times P(\Omega)$. Then we say that F satisfies the F-axiom if for any $(X,Y) \in P(\Omega) \times P(\Omega) \setminus F$ there is an $E \subseteq \Omega$ such that

(i) $X \subseteq E$ and $Y \not\subseteq E$;

(ii) if $(X',Y') \in F$ and $X' \subseteq E$ then $Y' \subseteq E$.

D-axiom: Let $D \subseteq P(\Omega) \times P(\Omega)$. Then we say that D satisfies the D-axiom if for any $(X,Y) \in P(\Omega) \times P(\Omega) \setminus D$ there is an $E \subseteq \Omega$ such that

(i) $X \cap E \neq \emptyset$ and $Y \cap E = \emptyset$;

(ii) if $(X',Y') \in D$ and $X' \cap E \neq \emptyset$ then $Y' \cap E \neq \emptyset$.

S-axiom: Let $S \subseteq P(\Omega) \times P(\Omega)$. Then we say that S satisfies the S-axiom if for any $(X,Y) \in P(\Omega) \times P(\Omega) \setminus S$ there is an $E \subseteq \Omega$ such that

(i) $X \cap E \neq \emptyset$ and $Y \not\subseteq E$;

(ii) if $(X',Y') \in S$ and $X' \cap E \neq \emptyset$ then $Y' \subseteq E$.

W-axiom: Let $W \subseteq P(\Omega) \times P(\Omega)$. Then we say that W satisfies the W-axiom if for any $(X,Y) \in P(\Omega) \times P(\Omega) \setminus W$ there is an $E \subseteq \Omega$ such that

(i) $X \subseteq E$ and $Y \cap E = \emptyset$;

(ii) if $(X',Y') \in W$ and $X' \subseteq E$ then $Y' \cap E \neq \emptyset$.

Theorem 1.1. (i) Let $F \subseteq P(\Omega) \times P(\Omega)$. Then F satisfies the $\mathfrak{f}$-axioms iff F satisfies the F-axiom.

(ii) Let $D \subseteq P(\Omega) \times P(\Omega)$. Then D satisfies the ϑ-axioms iff D satisfies the D-axiom.

(iii) Let $S \subseteq P(\Omega) \times P(\Omega)$. Then S satisfies the γ-axioms iff S satisfies the S-axiom.

Proof: (i) Suppose that F satisfies the F-axiom. Then

(F1) If $(A,A) \notin F$ then there is an $E \subseteq \Omega$ such that $A \subseteq E$ and $A \not\subseteq E$ which is a contradiction

(F2) If $(A,B)\in F$, $(B,C)\in F$ and $(A,C)\notin F$ then there is an $E\subseteq\Omega$ such that $A\subseteq E$ and $C\not\subseteq E$. Furthermore $(A,B)\in F$, $A\subseteq E$ imply $B\subseteq E$ and using $(B,C)\in F$, $C\subseteq E$ which is a contradiction.

The proof of (F3) and (F4) is analogous.

Suppose now that F satisfies the f-axioms. Let $(A,B)\in P(\Omega)\times P(\Omega)\setminus F$

Claim: There is an $E\supseteq A$ such that $(E,B)\in P(\Omega)\times P(\Omega)\setminus F$ and $E'\supset E$ implies $(E',B)\in F$.

$(\Omega,\Omega)\in F$ by (F1). Thus, by (F3) $(\Omega,B)\in F$ holds. $A\subseteq\Omega$ and $(A,B)\in P(\Omega)\times P(\Omega)\setminus F$, consequently there is an $E\subseteq\Omega$ which is maximal w.r. to the properties $(E,B)\in F$ and $E\supseteq A$.

This E clearly satisfies the restrictions of the Claim. Let $E\supseteq A$ which is guaranteed by the Claim. We state that E satisfies (i) and (ii) of the F-axiom. Namely by the choice of E, $A\subseteq E$ holds. By (F1) and (F3) $B\subseteq E$ implies $(E,B)\in F$. Thus we have $B\not\subseteq E$.

Let $(C,D)\in F$ and $C\subseteq E$. $D\not\subseteq E$ implies $E'=DUE\supset E$ and by the maximality of E $(E',B)\in F$ holds.

$(E,E)\in F$ by (F1), hence (F4) implies that $(E,E')\in F$. Now $(E,E')\in F$ and $(E',B)\in F$ and (F2) imply that $(E,B)\in F$ which is a contradiction.

(ii) Let $F=\{(A,B):(B,A)\in D\}$. Then by lemma 1.1 F satisfies the f-axioms iff D satisfies the ϑ-axioms. Hence, by (i), it is enough to show that F satisfies the F-axiom iff D satisfies the D-axiom.

Suppose that F satisfies the F-axiom. For $(A,B)\in P(\Omega)\times P(\Omega)\setminus F$ let $E(A,B)$ be such a subset of Ω that $A\subseteq E(A,B)$, $B\not\subseteq E(A,B)$ and if both $(A',B')\in F$ and $A'\subseteq E(A,B)$, then $B'\subseteq E(A,B)$. By the F-axiom such an $E(A,B)$ exists. By the definition of F whenever $(A,B)\in P(\Omega)\times P(\Omega)$ then $(A,B)\in P(\Omega)\times P(\Omega)\setminus F$ iff $(B,A)\in P(\Omega)\times P(\Omega)\setminus D$.

Now it is easy to chechk that for $(B,A)\in P(\Omega)\times P(\Omega)\setminus D$ $\Omega\setminus E(A,B)$ satisfies the D-axiom.

If D satisfies the D-axiom, then F satisfies the F-axiom; this can be shown by the same argument.

(iii) Suppose that S satisfies the S-axiom. Then the proof of the fact that S satisfies the γ-axioms is an easy modification of the proof of (i).

Suppose now that S satisfies the γ-axioms.
Let $(A,B)\in P(\Omega)\times P(\Omega)\setminus S$

Claim: There is an $a\in A$ and an $E\subseteq\Omega$
such that (a) $a\in E$;
(b) $(\{a\},E)\in S$ and
(c) $E'\supset E$ implies that $(\{a\},E')\notin S$.

If for any $a\in A$ we have $(\{a\},B)\in S$ then $(A,B)\in S$ by the repeated application of (S5). Hence there is an $a\in A$ such that $(\{a\},B)\notin S$. Now if for every $b\in B$ $(\{a\},\{b\})\in S$ holds then by the repeated application of (S4) we have $(\{a\},B)\in S$. Thus there is a $b\in B$ such that $(\{a\},\{b\})\notin S$.

By (S1) and (S3) there is an $E\subseteq\Omega$ such that $a\in E$, $(\{a\},E)\in S$ and E is maximal w.r. to this property. This E is appropriate for the Claim.

Let $E\subseteq\Omega$ and $a\in A$ guaranteed by the Claim. Then by (S3) we have $b\notin E$. Hence $A\cap E\neq\emptyset$ and $B\cap(\Omega\setminus E)\neq\emptyset$. Now let $(C,D)\in S$ such that $C\cap E\neq\emptyset$; let $c\in C\cap E$. Suppose that $D\cap(\Omega\setminus E)\neq\emptyset$; let $d\in D\cap(\Omega\setminus E)$. By (S3) we have $(\{c\},\{d\})\in S$ and by (S1) we have $(\{c\},\{c\})\in S$ $\cdot(\{a\},E)\in S$ implies that $(\{a,c\},\{c\})\in S$, by (S5). Hence (S3) implies that $(\{a\},\{c\})\in S$. Now $(\{a\},\{c\})\in S$, $(\{c\},\{d\})\in S$ and (S2) imply that $(\{a\},\{d\})\in S$. Thus by (S4) we have $(\{a\},E\cup\{d\})\in S$ which is a contradiction as $E' = E\cup\{d\}\supset E$.

Consequently the E guaranteed by the Claim demonstrates that S satisfies the S-axiom. □

It is worth to remark how can be find the full y-family - for $y\in\{f,d,s\}$ - generated by a given subset of $P(\Omega)\times P(\Omega)$ based on the Y-axiom. Let e.g. $y=f$ and let be given an $F'\subseteq P(\Omega)\times P(\Omega)$. Then the least full f-family containing F' is the following:

$$F = \{(A,B): A,B \subseteq \Omega \,\&\, (\forall E \subseteq \Omega)((A \subseteq E \,\&\, B \not\subseteq E) \rightarrow \rightarrow (\exists (A',B') \in F')(A' \subseteq E \,\&\, B' \not\subseteq E)\}.$$

§2. THE EQUALITY-SET

Definition 2.1. Let R be a relation over Ω. We define the equality set of R, $\&_R$ as follows. For $h,g \in R$ let $E(h,g) = \{a \in \Omega : h(a) = g(a)\}$ and let $\&_R = \{E(h,g) : h,g \in R \text{ and } h \neq g\}$.

Definition 2.2. Let $\mathbf{A}$ be a set system. Then $\mathbf{A}$ is a Δ-system if for any $A,B,C,D \in \mathbf{A}$ $A \neq B$ and $C \neq D$ implies that $A \cap B = C \cap D$.

Remark: It is easy to see that $\mathbf{A}$ is a Δ-system iff for any $A,B \in \mathbf{A}$ $A \neq B$ implies that $A \cap B = \cap \mathbf{A}$.

Theorem 2.1. (i) Let R be a relation over Ω and let h,f,g be different elements of R. Then $E(h,g)$, $E(h,f)$, $E(g,f)$ form a Δ-system.

(ii) Let $\& = \{E_{i,j} : 1 \leq i < j \leq k\}$ such that for each $1 \leq i < j < \ell \leq k$ $\{E_{i,j}, E_{i,\ell}, E_{j,\ell}\}$ is a Δ-system. Then there is a relation R over Ω with $\&_R = \&$.

Proof: (i) by symmetry it is enough to prove that $a \in E(h,g) \cap E(h,f)$ implies $a \in E(g,f)$. But this is trivial as $a \in E(h,g) \cap E(h,f)$ means both $h(a) = g(a)$ and $h(a) = f(a)$. Hence $g(a) = f(a)$ i.e. $a \in E(g,f)$.

(ii) We construct the rows of R by induction. Suppose that $n < k$, and the rows $h_1, \ldots h_n$ have been constructed so that for each $1 \leq i < j \leq n$ $E(h_i,h_j) = E_{i,j}$ holds. We construct h_{n+1} as follows:

$$h_{n+1}(a) = \begin{cases} h_i(a), & \text{if } a \in E_{i,n+1} \text{ for some } 1 \leq i \leq n; \\ \max (h_i(b) : b \in \Omega \,\&\, 1 \leq i \leq n) + 1 & \text{else.} \end{cases}$$

Then

(a) h_{n+1} is well-defined.

To prove this we have to show that $a \in E_{i,n+1} \cap E_{j,n+1}$ implies $h_i(a) = h_j(a)$. But this is obvious because $E_{i,j}$, $E_{i,n+1}$, $E_{j,n+1}$ from a Δ-system and the induction hypothesis hold for $i,j \leq n$.

(b) if $1 \leq i \leq n$ and $a \notin E_{i,n+1}$

then $h_i(a) \neq h_{n+1}(a)$.

Suppose first that $a \in E_{j,n+1}$ for some $1 \leq j < n+1$. Then, by (a) and by the definition of h_{n+1}, $h_{n+1}(a) = h_j(a)$ holds. Furthermore $a \notin E_{i,j}$ because $\{E_{i,j}, E_{j,n+1}, E_{i,n+1}\}$ is a Δ-system. Thus the induction hypothesis implies $h_i(a) \neq h_j(a)$, that is $h_i(a) \neq h_{n+1}(a)$.

If $a \notin \bigcup_{1 \leq j \leq n} E_{j,n+1}$ then we have $h_{n+1}(a) \neq h_i(a)$ by the definition of h_{n+1}. This completes the proof of (b).

Now by (a) and (b) it is clear that for $1 \leq i \leq n$ $E(h_i, h_{n+1}) = E_{i,n+1}$ and hence the induction step works. Let $R = \{h_1, \ldots h_k\}$. Then $\&_R = \&$ obviously holds. □

After Tehorem 2.1 there is a natural way to axiomatize full families of dependencies of any type. This follows next:

F'-axiom: Let $F \subseteq P(\Omega) \times P(\Omega)$. Then F satisfies the F'-axiom if there is a natural number k and an indexed set of subsets of Ω, $\{E_{i,j} : 1 \leq i < j \leq k\}$ such that

(i) If $(X,Y) \in P(\Omega) \times P(\Omega) \setminus F$ then there are $1 \leq i < j \leq k$ such that $X \subseteq E_{i,j}$ and $Y \not\subseteq E_{i,j}$.

(ii) If $(X,Y) \in F$, $1 \leq i < j \leq k$ and $X \subseteq E_{i,j}$ then $Y \subseteq E_{i,j}$.

(iii) For any $1 \leq i < j < \ell \leq k$ $\{E_{i,j}, E_{i,\ell}, E_{j,\ell}\}$ is a Δ-system.

D'-axiom: Let $D \subseteq P(\Omega) \times P(\Omega)$. Then D satisfies the D'-axiom if there is a natural number k and an indexed set of subsets of Ω, $\{E_{i,j} : 1 \leq i < j \leq k\}$ such that

(i) If $(X,Y) \in P(\Omega) \times P(\Omega) \setminus D$ then there are $1 \leq i < j \leq k$ such that $X \cap E_{i,j} \neq \emptyset$ and $Y \cap E_{i,j} \neq \emptyset$.

(ii) If $(X,Y) \in D$, $1 \leq i < j \leq k$ and $X \cap E_{i,j} \neq \emptyset$ then $Y \cap E_{i,j} \neq \emptyset$.

(iii) The same as (iii) of the F'-axiom.

S'-axiom: Let $S \subseteq P(\Omega) \times P(\Omega)$. Then S satisfies the S'-axiom if there is a natural number k and an indexed set of subsets of Ω, $\{E_{i,j} : 1 \leq i < j \leq k\}$ such that

(i) If $(X,Y) \in P(\Omega) \times P(\Omega) \setminus S$ then there are $1 \leq i < j \leq k$ such that $X \cap E_{i,j} \neq \emptyset$ and $Y \not\subseteq E_{i,j}$.

(ii) If $(X,Y) \in S$, $1 \leq i < j \leq k$ and $X \cap E_{i,j} \neq \emptyset$ then $Y \subseteq E_{i,j}$.

(iii) The same as (iii) of the F'-axiom.

W'-axiom: Let $W \subseteq P(\Omega) \times P(\Omega)$. Then W satisfies the W'-axiom if there is a natural number k and an indexed set of subsets of Ω, $\{E_{i,j} : 1 \leq i < j \leq k\}$ such that

(i) If $(X,Y) \in P(\Omega) \times P(\Omega) \setminus W$ then there are $1 \leq i < j \leq k$ such that $X \subseteq E_{i,j}$ and $Y \cap E_{i,j} = \emptyset$.

(ii) If $(X,Y) \in W$, $1 \leq i < j \leq k$ and $X \subseteq E_{i,j}$ then $Y \cap E_{i,j} \neq \emptyset$.

(iii) The same as (iii) of the F'-axiom.

Remark: Observe that the $E_{i,j}$-s in the F'-axiom are maximal dependent sets i.e. if $(X,Y) \in F$ and $X \subseteq E_{i,j}$ then $Y \subseteq E_{i,j}$.

Theorem 2.2. (i) Let $Y \subseteq P(\Omega) \times P(\Omega)$ and $Y \in \{F, D, S\}$. Then Y satisfies the Y-axiom iff Y satisfies Y'-axiom.

(ii) Let Ω be a finite set, $|\Omega|\geq 3$.
Then there is a $W\subseteq P(\Omega)\times P(\Omega)$ such that W satisfies the W-axiom and W doesn't satisfy the W'-axiom.

Proof: (i) Let first $Y=F$ and suppose that Y satisfies the F-axiom. Write $Y=F$.
For any $(X,Y)\in P(\Omega)\times P(\Omega)\setminus F$ take an $E(X,Y)\subseteq\Omega$ guaranteed by the F-axiom. List these $E(X,Y)$-s as $E_2,\ldots E_k$ the indexes begin with(2!). For $1<j\leq k$ let $E_{1,j}=E_j$ and for $1<i<j\leq k$ let $E_{i,j}=E_i\cap E_j$.
We claim that $\{E_{i,j}:1\leq i<j\leq k\}$ demonstrates that F satisfies the F'-axiom.
The requirement (i) of the F'-axiom holds by $\{E_2,\ldots,E_k\}\subseteq\{E_{i,j}:1\leq i<j\leq k\}$.
We left to the reader to check that (ii) holds too.
To prove (iii) of the F'-axiom let $1\leq i<j<\ell\leq k$. We distinguish two cases:

(a) $i=1$. Then $E_{i,j}=E_j$; $E_{i,\ell}=E_\ell$ and $E_{j,\ell}=E_j\cap E_\ell$. Thus the intersection of any two members of $\{E_{i,j};E_{i,\ell};E_{j,\ell}\}$ is $E_j\cap E_\ell$. This means that $\{E_{i,j};E_{i,\ell};E_{j,\ell}\}$ is a Δ-system.

(b) $1<i$. Then $E_{i,j}=E_i\cap E_j$; $E_{i,\ell}=E_i\cap E_\ell$ and $E_{j,\ell}=E_j\cap E_\ell$. Thus the intersection of any two members of $\{E_{i,j};E_{i,\ell};E_{j,\ell}\}$ is $E_i\cap E_j\cap E_\ell$. This means that $\{E_{i,j};E_{i,\ell};E_{j,\ell}\}$ is a Δ-system.

If Y satisfies the F'-axiom then Y obviously satisfies the F-axiom.

Now let $Y=D$ and suppose that Y satisfies the D-axiom. Write $Y=D$.
For any $(X,Y)\in P(\Omega)\times P(\Omega)D$ take an $E(X,Y)\subseteq\Omega$ guaranteed by the D-axiom. List these $E(X,Y)$-s as $E_1,\ldots E_k$.
For $1\leq i\leq k$ let $E_{2i-1,2i}=E_i$ and if $1\leq i<j\leq 2k$ and $E_{i,j}$ is still undefined then let $E_{i,j}=\emptyset$.

It is easy to see that $\{E_{i,j}: 1\leq i<j\leq 2k\}$ shows the D'-axiom to hold for **D**.

If D satisfies the D'-axiom then it trivially satisfies the D-axiom.

The case Y=S is an easy modification of the proof worked in the case Y=F.

(ii) For the sake of simplicity suppose that $\Omega=\{a,b,c\}$. (In the general case pick two different elements of Ω, a,b. The role of {c} will be played by $\Omega\setminus\{a,b\}$.)

Let $W=\{(A,B)\in P(\Omega)\times P(\Omega)$: $A\subseteq\{a\} \rightarrow a\in B$ and $A\subseteq\{b\} \rightarrow b\in B\}$

Then W satisfies the W-axiom while if $(A,B)\in P(\Omega)\times P(\Omega)\setminus W$ then either ($A\subseteq\{a\}$ and $a\notin B$) or ($A\subseteq\{b\}$ and $b\notin B$. For (A,B) $E=\{a\}$ taken in the first case and $E=\{b\}$ in the second shows the W-axiom to hold.

We claim that W doesn't satisfy the W'-axiom. Suppose indirectly that $\&=\{E_{i,j}: 1\leq i<j\leq k\}$ is a system that shows the W'-axiom to hold for W. Then

(1) $\{a\}\in\&$ and $\{b\}\in\&$

while $(\{a\},\Omega\setminus\{a\})\in P(\Omega)\times P(\Omega)\setminus W$ and $(\{b\},\Omega\setminus\{b\})\in P(\Omega)\times P(\Omega)\setminus W$ hold.

(2) $\emptyset\notin W$ and $\{c\}\notin W$

while $(\emptyset,\Omega)\in W$ and $(\{c\},\Omega\ \{c\})\in W$ hold.

By the "allocation" of {a} and {b}, we distinguish two cases:

(a) $\{a\}=E_{i,j}$ and $\{b\}=E_{i,\ell}$

Then $\{E_{i,j}; E_{i,\ell}; E_{j,\ell}\}$ is a Δ-system, that is $E_{j,\ell}=\emptyset$ or {c} which contradicts (2).

(b) $\{a\}=E_{i,j}$ and $\{b\}=E_{\ell,m}$, where $|\{i,j,\ell,m\}|=4$.

Now we are interested but in $E_{i,j}$; $E_{i,\ell}$; $E_{i,m}$; $E_{j,\ell}$; $E_{j,m}$ and $E_{\ell,m}$, thus we may suppose that i=1, j=2, ℓ=3 and m=4.

Investigate what may be $E_{1,3}$.

The cases $E_{1,3} = \{a\}$ or $\{b\}$ arise to (a).

$E_{1,3} \neq \{c\}$ and $E_{1,3} \neq \emptyset$ by (2).

$E_{1,3} \neq \{b,c\}$ while $\{E_{1,2};E_{1,3};E_{2,3}\}$ is a Δ-system hence $E_{1,3} = \{b,c\}$ implies $E_{2,3} = \emptyset$ contradicting (2). Now it is clear that $a \in E_{1,3}$. Thus $a \in E_{2,3}$, while $\{E_{1,2};E_{1,3};E_{2,3}\}$ is a Δ-system.

$\{E_{2,3}; E_{2,4};E_{3,4}\}$ is a Δ-system, hence a $E_{2,4}$, that is $E_{2,4} \subseteq \{b,c\}$.

$E_{2,4} \neq \emptyset$ and $E_{2,4} \neq \{c\}$ by (2) and $E_{2,4} \neq \{b\}$ by (a). Hence $E_{2,4} = \{b,c\}$.

$\{E_{2,3}; E_{2,4};E_{3,4}\}$ is a Δ-system, hence $b \in E_{2,3}$.

Finally $E_{1,3} = \{a,c\}$ while $E_{2,3};E_{1,2};E_{1,3}$ form a Δ-system.

Now $\{E_{1,3};E_{1,4};E_{3,4}\}$ is a Δ-system and $E_{1,3} \cap E_{3,4} = \emptyset$ and $E_{1,3} \cup E_{3,4} = \Omega$, hence $E_{1,4} = \emptyset$ which contradicts (2). □

Remark: Theorem 2.2 demonstrates the difference between the weak dependency and the rest.

Theorem 2.3. Let $Y \subseteq P(\Omega) \times P(\Omega)$ satisfy the Y'-axiom for some $Y \in \{F,D,S,W\}$. Then there is a relation R over Ω with $Y = Y_R$. Conversely if R is a relation over Ω then Y_R satisfies the Y'-axiom.

Proof: Let $\& = \{E_{i,j} : 1 \leq i < j \leq k\}$ show that Y satisfies the Y'-axiom. Then the requirement (iii) of the Y'-axiom and Theorem 2.1 (ii) imply that there is a relation R over Ω such that $\&_R = \&$. By the Y'-axiom it is obvious that $Y = Y_R$. Conversely, if R is a relation over Ω, then writing $R = \{h_1, \ldots h_k\}$, $E_{i,j} = E(h_i, h_j)$; $\{E_{i,j} : 1 \leq i < j \leq k\}$ shows that Y_R satisfies the Y'-axiom. □

§3. COMBINATORIAL RESULTS

Definition 3.1. Let $\mathbf{F}$ be a full f-family and let $A \subseteq \Omega$. Then A is a candidate key for $\mathbf{F}$ if $(A, \Omega) \in \mathbf{F}$ and for any $A' \subset A$ $(A', \Omega) \notin \mathbf{F}$ holds. Let R be a relation over Ω, then the set of candidate keys of R is the set of candidate keys of $\mathbf{F}_R$.

Let $\mathbf{C}$ denote the set of candidate keys of $\mathbf{F}$. Then $\mathbf{C}$ is a Sperner system i.e. $(\forall A, B \in \mathbf{C})(A \subseteq B \rightarrow A = B)$.
We deal with the following question of [11]:

(*) What is the largest number $r(n)$ of rows that is needed for some $\mathbf{C} \subseteq P(\Omega)$ being the set of candidate keys of a relation over Ω with $r(n)$ rows, where $|\Omega| = n$ and $\mathbf{C}$ is a Sperner system?

In [11] it is shown that for any Sperner system there is a relation with this system as its set of candidate keys and that

$$\sqrt{2^{\binom{n}{[n/2]}}} \leq r(n) \leq 2^{\binom{n}{[n/2]}}.$$

We give sharper estimations for $r(n)$.

Theorem 3.1. $$\frac{1}{n^2} \binom{n}{[n/2]} < r(n) \leq \binom{n}{[n/2]}^{+1}.$$

Proof: First we prove the upper bound.
Let $\mathbf{C} \subseteq P(\Omega)$ be a Sperner system. Let $\mathbf{B}$ consist of the maximal sets that do not contain members of $\mathbf{C}$. Let the members of $\mathbf{B}$ be $B_2, \ldots B_k$. For $1 < j \leq k$ let $E_{1,j} = B_j$ and

for $1<i\leq j\leq k$ let $E_{i,j} = B_i \cap B_j$. Then $\{E_{i,j}: 1\leq i<j\leq k\}$ satisfies the requirements of the Theorem 2.1 (ii), hence there is a relation R over Ω with k rows such that $\mathfrak{E}_R = \{E_{i,j}: 1\leq i<j\leq k\}$. Then obviously $\mathbf{C}$ is the set of candidate keys of R. It is trivial that $\mathbf{B}$ is a Sperner system, and thus $|\mathbf{B}| \leq \binom{n}{[n/2]}$ that is

$k \leq \binom{n}{[n/2]} + 1$.

Now let we seen the lower bound. We start with two trivial observations.

1. Let R be a relation over Ω with r rows. Then there is a relation R' over Ω such that R' uses no more than r symbols and $\mathfrak{E}_R = \mathfrak{E}_{R'}$.
2. Let R be a relation over Ω with r rows and let $r'>r$. Then there is a relation R' over Ω with r' rows such that $\mathfrak{E}_R = \mathfrak{E}_{R'}$. (Now we allow identical rows.)

By 1. and 2. the number of Sperner systems which may be represented as sets of candidate keys of a relation with r rows is no more then $r^{r.n}$. Hence

$$r(n)^{r(n).n} > 2^{\binom{n}{[n/2]}} \quad \text{which implies}$$

$$r(n) > \frac{1}{n^2}\binom{n}{[n/2]}. \qquad \square$$

If $\mathbf{B}$ is a Sperner system and R is a relation such that $\mathbf{B} \subseteq \mathfrak{E}_R \subseteq \{\cap \mathbf{B}': \mathbf{B}' \subseteq \mathbf{B}\}$ then we can define two graphs on the set of rows of R as follows:

1. the $\mathbf{B}$-graph of R is G_R where the vertices of G_R are the rows of R and two rows are connected by an edge if and only if their equality-set is an element of $\mathbf{B}$
2. the colored graph of R is the complet graph on the set of rows of R with the colour E(f,g) on the edge {f,g}.

The $\mathbf{B}$-graph of R has the following property: if G_R is disconnected, then there is a relation R' such that the number of

rows of R' is less than that of R and $\mathbf{B} \subseteq \&_{R'} \subseteq \{\cap \mathbf{B}' : \mathbf{B}' \subseteq \mathbf{B}\}$. The colored graph of R contains no circuit the edges of which have the same colour except exactly one.

These two observations may be useful to make an algorithm to find the minimal relation for Sperner systems.

The estimation for $r(n)$ in Theorem is not sharp. If $\mathbf{B} = \{X \subseteq \Omega : |X| = [\frac{n}{2}]\}$, then there is a relation R such that $\mathbf{B} \subseteq \&_R \subseteq \{\cap \mathbf{B}' : \mathbf{B}' \subseteq \mathbf{B}\}$ and the number of rows of R is the least natural number greater than

$$\frac{1}{2}\binom{n}{[n/2]} + \sqrt{2\binom{n}{[n/2]}}.$$

It is natural to ask the following analogon of (*): What is the largest number $R(n)$ of rows that is needed to represent a relation with $\mathbf{F}$ as the set of functional dependencies of it for an $\mathbf{F} \subseteq P(\Omega) \times P(\Omega)$ where $|\Omega| = n$ and $\mathbf{F}$ is a full f-family.

By the proof of Theorem 2.2 (i) it is obvious that $R(n) \leq$ (the maximal number of subsets of Ω such that the intersection of any two of them is not a third). Thus, by a theorem of D.Kleitman [13], $R(n) \leq c.\binom{n}{[n/2]}$ where $c = 3/2$. Z.Füredi and J.Pach have shown, that this number is less then $(1+(c.\log n)/n)\binom{n}{[n/2]}$. It is trivial that $r(n) \leq R(n)$.

Lastly we give the combinatorial characterization - according to §0 - of the sets which are of minimal cardinality with respect to the property that they imply all the dependencies of a given full f-family.

We need some definitions and a lemma.

Definition 3.2. Let $\mathbf{M} \subseteq P(\Omega)$.

(i) We say that $\mathbf{M}$ has the intersection property if for any $\mathbf{M}' \subseteq \mathbf{M}$ $\cap \mathbf{M}' \in \mathbf{M}$ holds.

(ii) An $M \in \mathbf{M}$ is irreducible if $M \neq \cap\{M' \in \mathbf{M} : M \subset M'\}$ (recall that $\subset$ means strict inclusion)

(iii) An $\mathbf{N} \subseteq \mathbf{M}$ generates $\mathbf{M}$ if $\mathbf{M} = \{\cap \mathbf{N}' : \mathbf{N}' \subseteq \mathbf{N}\}$.

Lemma 3.1 Let $\mathbf{M}$ have the intersection property and let $\mathbf{N} = \{M \in \mathbf{M}: M \text{ is irreducible}\}$. Then an $\mathbf{N}' \subseteq \mathbf{M}$ generates $\mathbf{M}$ iff $\mathbf{N} \subseteq \mathbf{N}'$.

Proof: The following proof is standard in lattice theory. If $\mathbf{N}'$ generates $\mathbf{M}$, then $\mathbf{N} \subseteq \mathbf{N}'$ is obvious. For the converse we have to prove that $\mathbf{N}$ generates $\mathbf{M}$. Suppose indirectly that there is an $X \in \mathbf{M} \setminus \mathbf{N}$ such that $X \neq \cap\{Y: Y \in \mathbf{N} \& X \subseteq Y\}$. Let X be of minimal cardinality with respect to this property. $X \notin \mathbf{N}$ means that $X = \cap\{Y: Y \in \mathbf{M} \& X \subset Y\}$, hence $X \subset Y$ implies that there is an $\mathbf{N}_Y \subseteq \mathbf{N}$ such that $Y = \cap \mathbf{N}_Y$. Let $\mathbf{N}_X = \cup\{\mathbf{N}_Y: X \subset Y \text{ and } Y \in \mathbf{M}\}$.

Then $\mathbf{N}_X \subseteq \mathbf{N}$ and $X = \cap \mathbf{N}_X$ which is a contradiction. □

Remark: Observe that the proofs of Theorems in [2] are essentially our proof of Lemma 3.1.

Corollary: If $\mathbf{M}$ has the intersection property then there is exactly one $\mathbf{N} \subseteq \mathbf{M}$ which generates $\mathbf{M}$ and has minimal cardinality.

Theorem 3.2. Let $\mathbf{F}$ be a full f-family, let $\mathbf{B}$ be the set of maximal dependent set for $\mathbf{F}$ and let $\mathbf{C}$ be the set which generates $\mathbf{B}$ and has minimal cardinality (in [1] there is shown that $\mathbf{B}$ has the intersection property).

Then for any $\mathbf{F}' \subseteq \mathbf{F}$ we have the following:

$\mathbf{F}'$ implies all the dependencies of $\mathbf{F}$ and $\mathbf{F}'$ has minimal cardinality with respect to this property

if and only if

for any $C \in \mathbf{C}$ there is an $A_C \subseteq \Omega$ such that $\mathbf{F}' = \{(A_C, C) : C \in \mathbf{C}\}$.

We left the easy proof of the Theorem to the reader. We think that it is interesting to compare Theorem 3.2 with the Theorem on pp. 16 of [2].

R E F E R E N C E S

[1] ARMSTRONG,W.W., Dependency structures of data base relationships, Information Processing 74, North-Holland Publ. Co., (1974) 580-583.

[2] ARMSTRONG,W.W., On the generation of dependency structures of relational data bases, Publication ≠ 272, Universite de Montréal.(1977).

[3] BEERI,C. - FAGIN,R. - HOWARD,J.H., A complete aximatization for functional and multivalued depencies in database relations, Proc. ACM SIGMOND Int.Conf. an Management of Data, Toronto(1977) 47-61.

[4] BÉKÉSSY,A. - DEMETROVICS,J., Contribution to the theory of data base relations, Discrete Math. 27 (1979) 1-10.

[5] BÉKÉSSY,A. - DEMETROVICS,J. - HANNÁK,L. - FRANKL,P. - KATONA,G., On the number of maximal dependencies in a data base relation of fixed order. Discrete Math. 30 (1980) 83-88.

[6] CODD,E.F., A relational model of data for large shared data banks, Comm. ACM. 13 (1970) 377-387

[7] CODD,E.F., Further normalization of the data base relational model, Courant Computer Science Symposia 6 Data Base System, Prentice Hall, Englewood Cliffs, N.J., (1971) 33-64.

[8] CZÉDLI, G.: Függőségek relációs adatbázis modellben Alk.Mat.Lapok (1980)

[9] DELOBEL,C., Normalization and hierarchical Dependencies in the Relational Data Model. ACM Trans. an Database Sys. 3 (1978) 201-222

[10] DELOBEL,C. - CASEY,R.G., Decomposition of a Data Base and the Theory of Boolean Switching Functions, IBM J. Res.Develop. September (1973) 374-386.

[11] DEMETROVICS,J., Candidate keys and antichains, SIAM J. on Algebraic and Discrete Methods 1(1980) 92.

[12] DEMETROVICS,J., On the equivalence of candidate keys with Sperner systems, Acta Cybernetica 4 (1979) 247-252.

[13] FAGIN,R., Multivalued dependencies and a new normal form for relational data-bases, ACM Trans. Database Sys. 2(1977) 262-278

[14] KLEITMAN,D., On a problem of Erdős, Proc. of the Am. Match. Soc. 18(1966) 139-141

[15] MENDELZON,A.O., On axiomatizing multivalued dependencies in relational databases, J. Ass. for Comp. Sci. 26 (1979) 37-44.

[16] RISSANEN,J., Independent components of relations, ACM Trans. Database Sys., 2(1977) 317-325

[17] SPERNER,E., Eine Satz über Untermengen einer endlichen Menge, Math.Z. 27 1928 544-548.

[18] YU,C.T. - JOHNSON,D.T., On the complexity of finding the set of candidate keys for a given set of functional dependencies, Information Processing Letters 5(1976) 100-101

Author's address:

J.Demetrovics, Gy.Gyepesi: Computer and Automation Institute of Hungarian Academy of Sciences H 1502 Budapest, XI. Kende u. 13-17.

MTA Számitástechnikai és Automatizálási Kutató Intézete, Tanulmányok 133/1982 Proc. of RG-11, KNVVT

Some Remarks on Statistical Data Processing

Pál Kerékfy

Computer and Automation Institute
Hungarian Academy of Sciences
Budapest P. O. Box 63, H-1502

1. INTRODUCTION

The so-called "software problem" or "software crisis" is the most important matter at issue in computer science. Several papers are devoted to discuss different aspects of the crisis /see e.g. [8] or [16]/.

The author's contribution to the solution of the problem is a knowledge-based program generator system. We were inspired by the experiences showing that software demands of certain typical statistical procedures are not satisfied. The most important examples are processing /data checking, transformation/, input-output, data storage and collecting informations on the data /evaluation of frequencies, cumulation/. These tasks give rise to difficulties mainly in large and complicated statistical investigations. With our work started in the 70's we wished to obtain results just in this field.

Our first attempts were concluded from the Hungarian Hospital Morbidity Study [3] producing a simple statistical inquiry system. The method of program generating was applied in this system, and it proved to be useful. Based on our experiences, we constructed the program generator system GENERA.

System GENERA [12] was developed for the extension of a simple program generating method used in an earlier system, i.e. SIS77. It gives assistance to a program generator technique that makes the programming and usage of optimal-performance procedures possible. After generating the program, GENERA provides meand for compiling and executing it. The source program can contain parametrized macros, moreover the whole program /or job/ can be considered as a parametrized unit that can be run with

different parameters depending on the present usage requirements. An improved version of system SIS77, system SIS79/GENERA utilizes program GENERA [13].

While processing large and complicated data sets, beyond the problem of selecting proper software tools interesting mathematical /optimization/ problems arise. They can emerge while designing the codes used, sampling, designing the processes and structure of data storage.

Data checking, transformation and, in general, analysis of functions providing control, transformation or selection of the sample are questions of great importance. These tasks require /in the case of large and complicated system/ modelling of strange functions and convenient description of large code tables.

FORTRAN is a language that is widely used in writing programs for statistical data processing, some well-known systems /such as BMDP or SPSS/ utilize it. Present implementations of GENERA have FORTRAN as an optional host language /beside PL/1/. Data description and I/O operations of FORTRAN are sometimes inconvenient and slow. This fact inspired us to work out some procedures for input-output, data description and storage in systems SIS77 and SIS79.

In the following, systems SIS77 and SIS79 will be described using the Hungarian Hospital Morbidity Study as an example. The possibilities of solving the problems raised in the introduction of the paper will be discussed. Matters in computer science and mathematical statistics connected with the topic will be dealt as well.

2. HUNGARIAN HOSPITAL MORBIDITY STUDY AND SYSTEM SIS79/GENERA

In Hungary, representative hospital morbidity studies have been in progress /including each hospital and department/ since 1972. Data of the inpatients are collected yearly with a sampling rate ranging from 1o to 5o percent. It amounts to information on 200 to 600 thousand patients per year.

The problem was interesting for us as processing of a rather large sample /600 thousand records, 60 to 80 million bytes/ was to be accomplished on a comparatively small machine. The requirements were rather complicated and subject to modifications from time to time. At first, the machine used was a CDC-3300 having 64 Kwords of memory with two 8 Mbyte disc units and two or three tape units available. The machine was overloaded so we could run small jobs /some minutes of CPU time/ only. Consequently, jobs utilizing the total sample were to be run rarely. It became necessary to examine questions concerning the strategy of data processing. On the other hand, the data set was to be divided and compressed. The running time /and other resources as well: memory, disc, tape/ of the job was to be minimized.

Later we got access to higher-capacity machines [14] /a HwB 66/60 or two HwB 66/20's with 100 Mbyte discs and 256 Kwords of memory/. The problem of capacity became less important. But taking into account the requirements of conversational processing and the aspiration to faster turn-around in batch processing /and the expenses as well/ optimization of storage and time was expedient.

One method to achieve shorter run-time was used while creating statistical tables. They were created from frequencies and cumulated values instead of the original data. In this way statistical tables are obtained in some seconds, practically independently of the sample size [14]. Let us note here that the Hospital Morbidity Study required descriptive statistics mainly: tables to be produced contained frequencies, cumulated values and some simple rates /e.g. morbidity rate, etc./ and fundamental characteristics of the distribution /as mean, standard deviation, range/. The solution of problems of mathematical statistics needs frequencies and cumulated values /sums, quadratic sums, sums of products/ /see the statistical

literature [11]/. Then these values can be processed e.g. by SPSS programs.

The program generating technique /based on earlier experiences/ was consistently applied in system SIS77 developed on HwB 66's [21]. In its improved version /in SIS79/GENERA / this technique was developed further [13]. The task of generating was placed under control of a general purpose system /GENERA/. It improved the integrity and efficiency of the system. System GENERA and the possibilities provided by statistical system SIS79 will be dealt with in later sections.

Coming back to the Hungarian Hospital Morbidity Study we can state that our statistical system /SIS77, SIS79/ is able to provide quick access to data and detailed analysis even for individual researchers. It is made possible by the fact that even in the case of large mass of data and complicated conditions the system needs modest resources only. We can mention as an example the COMECON-project on juvenile hypertension coordinated by the National Cardiology Institute that was successfully accomplished by systems SIS77 and SIS79 [13], [14].

Among the above discussions on strategy we emphasize the following. In statistical tasks /especially in large and complicated systems/ the method of sequential processing is suitable. Sequential processing is similar to sequential sampling [23] known from mathematical statistics. It produces a more and more widening sphere of information depending on the information obtained before. But compared to sequential sampling it does not mean an increasing amount of information of the same kind; in this case the kind of information is subject to change as well. Users /doctors, economists, etc./ first receive simple, easy-to-survey data /tables, graphs, descriptive statistics/. The more and more detailed questions are based on the information obtained

earlier and can optionally be answered on the base of a widening population. /That is a wider subset of the data set./ In this way needless information is not to be gathered, simple relations are enlightened immediately and the user gets an overall picture of the sample investigated. This method provides means for obtaining more valuable information from the data available.

Determination of code values for data is another interesting and important problem in data oricessing. It may require mathematical statistical investigations as well as representative sampling. One of the problems in the hospital morbidity studies was producing a reliable identifier for patients. A comparatively short, easy-to-code identifier was required with negligible probability of accidental coincidence /wrong identification/. The task of selecting the representative sample was a problem of similar complexity. Our experiences showed that a sampling based on the birthday of inpatients is quite uniform [3]. Rate of multiple hospitalized persons showed little deviations only. Usually, representative sampling from individuals of multiple occurences is a complex matter requiring complicated mathematical investigations [9].

Besides, multiple hospitalized persons and inpatients having multiple diagnoses require a file organization different from usual statistical data bases. The elements to be examined are not the original records /hospitalized cases/. New basic elements /one multiple hospitalized person or one diagnosis/ are to be constructed. Problems of this kind are directly connected to data bases /to the relational data models [1], [4] especially/.

3. PROGRAM GENERATING

3.1 Introduction

The pressing matter of "software crisis" has not gone unrecognized in the recent years and a large number of automated methods and design principles has been developed.

The two basic approaches are the very high level languages and the knowledge-based systems, and they are well-known in the literature, see e.g. the bibliography of [10].

The software development technique to be introduced here is especially useful in large and complicated tasks that are to be solved by little efforts. Concerning its aims and methods it is close to the knowledge-based-system approach.

3.2 General description

System GENERA is a system to build generator programs having subsystems. Subsystems can have a set of parameters, they are given value by unified and flexible methods. A generator system based on GENERA has a predefined host language /or a set of host languages such as FORTRAN or PL/1 /. Text to be processed consists of host language statements and GENERA directives. The former ones become statements of the generated program whithout any modification. On the other hand, the appropriate text generated by the designated subsystem replaces the directive.

The term "host language" does not imply that the object to be generated has to be a program. It can be a text of any kind. Obviously, this freedom concerns requirements of GENERA itself. Host language is defined before a generator program is established. Then the language is fixed, and the system generates statements of that language.

The utilization of a high-level computer language for the purpose of macro description can widen the sphere of the tasks that can be solved by the macro expansion routines.

3.3 Structure of a system built up on GENERA

A system based on GENERA integrates any number of generator procedures to make a precompiler. These procedures from subsystems of the generator program and are

called into execution by entering a directive onto the sourec file. Detecting a directive, control is passed to the main entry point of the subsystem to read in the parameters. Then the subsystem is executed. Having completed its function, the subsystem returnes control to the main program to continue processing of the source file.

Generator subsystems are not part of system GENERA itself. They form the specific generator system constructed on the base of GENERA. However, there are some predefined GENERA directives calling certain procedures that are integrate part of any GENERA installation. They perform a number of prgram control functions.

OPTION is a subsystem of program control. It can be executed as the first step of a GENERA run and initializes some global variables of system to achieve a non-standard handling of source lines. By a proper choice of parameter values the user gains control over the structure and contents of output information.

A preprocessor subsystem /PREP/ is contained in batch oriented versions of GENERA. This is a subsystem that cannot be called in by the user directly, and is always executed prior to any other functions of GENERA. Each line of input is examined, lines containing directives or parameters are checked. Statistics of the recognized directives are collected, and the unrecognized ones are reported. Then the parameters are tested if they meet the rules defined for the subsystem. Having found an error, the run is terminated abnormally at the end of preprocessor phase. The preprocessor can detect some kinds of syntax errors before generating, thus needless execution of generator functions can be avoided. The preprocessor performs some transformations on parameter descriptions to provide an interface between user-oriented description

and the program requirements. There is a genrator procedure to generate the input phase of the subsystems. The input phase is activated by the preprocessor and it reads in the parameters specified in a very convenient method that is an extension of the PL/1 GET DATA and the FORTRAN NAMELIST input format. The parameters are checked and converted to the form of internal storage and stored in memory. At the time of execution of the subsystem it receives a pointer to the memory block containing its parameters. This method makes both the programming of subsystems and definition of parameters very convenient.

A simple macro generator is implemented in GENERA to provide means for definition and usage of macros. It has a library containing system macros defined by the system programmers and processes temporary macros defined by the user during the GENERA run. The macros can contain host language statements and generator directives as well.

As GENERA processes a number of input files /primary input containing host language program, directives and parameters; secondary input file containing any structured data for subsystems; job generator /JOBGEN/ input file describing non-standard job setup/ an Initial File Conversion subsystem /IFCS/ accompanies the system. IFCS builds up the input files from a single input file /MIXEDIN/ and it can include some additional features /such as selecting given disc files or tapes as parts of input file/ depending on the possibilities provided by the operating system.

4. STATISTICAL INFORMATION SYSTEM SIS79/GENERA

Systems SIS77 and SIS79/GENERA have been mentioned before. The most important procedures of SIS79 will be

presented here.

4.1 Data Transmission and Conversion

In a system to handle a great mass of data, efficiency of input-output operations is important. We developed a pair of I/O statements / LECTOR, SCRIPTOR/ to perform these operations. They are given the record structure /name, length and type of each field/, and a program-fragment is generated to read or write the annotated variables.

The example in the figure shows an input directive LECTOR. The meaning of the set PARAM goes without saying. The set DESCR gives format for reading record named PATIENT /COBO-style level numbers and FORTRAN format items are used/.

Procedures to generate I/O operations are needed in some systems because high-level languages analyse format specifications in run time. Formats are usually not changed while running the program, so run-time evaluation is not needed. However, compilers do not translate the format items to machine code.

Our input procedure generates a set of host-language statements to read the record "as-is" /without any conversion/ and to select and convert values of variables using efficient character-handling routines. Hence, format items are evaluated in compile-time instead of run-time. A large amount of processor time can be saved if there are I/O statements frequently used in the program. The method is especially useful in FORTRAN programs. The HwB version of GENERA was written in FORTRAN and the experiences showed that the processor time used for reading the same type of records for some hundred thousand

```
$OPTION
 $PARAM LIST='ERROR',SYSTEM='FORTRAN'$
       INTEGER AGE,HYEAR,BYEAR,SEX,PROFS,
      *CODE,MAINCD,SUBCD,ERROR(1)
 1     CONTINUE
$LECTOR
 $PARAM FC=5,END=500,ERR=600$
 $DESCR PATIENT
        1 NAME 30X
        1 BYEAR I4
        1 SEX I1
        1 COUNTY 2X
        1 PROFS I4
        1 HDATE
          2 HYEAR I4
          2 HMONTH 2X
          2 HDAY 2X
        1 CODE I4
          2 MAINCD I3
          2 SUBCD I1
 $
        AGE = HYEAR - BYEAR
$GRAPH
 $PARAM GRAPH='AGE CODING',DATANA='AGE',
NEWDAT='CDAGE',SACKNO=1,LEVELS=1,
UPPBOU=100$
        IF (NUMERR.NE.0) GOTO 100
$GRAPH
 $PARAM GRAPH='CONTROL',DATANA='CDAGE',
'SEX','MAINCD','SUBCD',SACKNO=34,LEVELS=
1,10,15,8,UPPBOU=10,10*10,896,788,
10*999,618,528,496,2*7,6*9,
LOWBOU(2)=10*1,LOWBOU(15)=125$
        IF (NUMERR.NE.0) GOTO 101
        GOTO 1
100     WRITE (6,10) ERROR(1)
10      FORMAT (' ERROR IN AGE:',I4)
        GOTO 1
101     WRITE (6,11) ERROR(1)
11      FORMAT (' ERROR:',I4)
        GOTO 1
600     WRITE (6,12)
12      FORMAT (' READ ERROR')
        GOTO 1
500     STOP
        END
```

Figure 1. Sample input of SIS79

times is reduced to ten percent of the original value.

4.2 Compressed Binary Storage

Data storage can be a problem of great importance in some statistical systems. Let us see the following example. A large amount of data is to be stored on mass storage devices. It is known that data set contains numbers of small values. These numbers can be described by one or two decimal digits but they are used frequently and character form requires a conversion to be performed each time the data are read or written. On the other hand, data stored in binary form can be read or written without conversion but in this case each number requires a full word of storage. /It is right for word oriented machines only./ We should find a method that is efficient in both means. That is, it should provide a fast conversion and the data should not occupy superflouos staorage space.

The compressed binary representation used in our system reduces the storage space required while processor time used is not increased significantly. It is achieved by compressing length of binary form to the number of bytes required to contain the greatest allowable value of the variable. The compressed binary read and write procedures generate a program fragment performing I/O operation and compression or decompression.

4.3 Data Transformation and Graph Representation of Functions

Data preparation tasks involving transformations /coding, analysis of functions/ are included in this group. To perform these tasks we have to descibe the transformation procedure itself. It does not cause any difficulty in the case of functional dependencies defined by simple formulae.. On the other hand, code-tables can be

extremely large, larger than the total amount of core memory available on the machine used. Description, control and storage of these tables can cause hardly resolvable problems. One of the installed generator system based on GENERA, system SIS79, involves a certain storage method especially designed to be used in generated programs.

Using this method, functions or transformations defined by code-tables are described in the form of a hierarchical graph [13], [14]. This graph is divided into levels corresponding to the arguments of the function. A level contains one or more subtable controlling values of the variable belonging to the level. The subtables are subscipted by the value of the argument to get the value of the function or the next subtable pointer. Being empty parts or identical segments included in the original code-table, this method can provide a significant reduction of storage required for the table. Moreover, an efficient program can be generated to read and analyse the graph. While the necessary storage capacity is radically reduced /e.g. in a system used by the Health Service, tables based on the international code system of disease /WHO-codes/ were reduced to five percent of the original size/, compute time did not increase essentially compared to the time required by the method using a unique large table of values. In this case the method of program generating proved to be very useful as the structure of a general program to analyse a graph of any kind is extremely complicated. On the contrary, the generated program is of clear structure and the generator program is rather simple.

Reduction of storage and run-time contains several interesting problems of graph theory and finite projective geometry, see e.g. [15].

The previous figure contains two consequtive GRAPH directives. The first, "AGE CODING", codes variable AGE to variable CDAGE using one code-table /SACKNO=1, LEVELS=1/. Maximal value allowed for AGE is 100 /UPPBOU=100/. Variable NUMERR is used for signaling errors. The second procedure "CONTROL", checks variables CDAGE, SEX, MAINCD and SUBCD using a graph of 4 levels and 34 elementary tables.

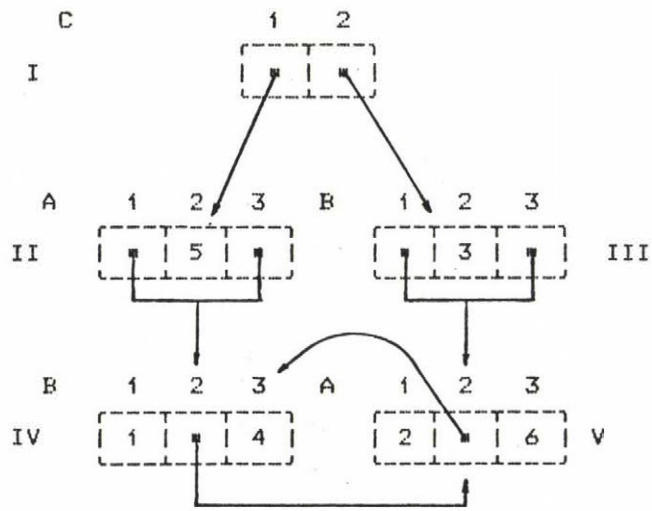
The tables are filled up by a general procedure contained in systems SIS77 and SIS79. Several advantages are obtained using this routine: the method applied to fill up the tables is the most compavt and comfortable one, appropriate security is provided by the syntax analysis of table descriptions and detailed error messages. This subroutine provides means for a quick and easy calculation of some multivariate functions as well. We demonstrate the method of constructing a graph in figures 2 to 4 using a very simple function. Table generating statements in figure 4 contains /from the left to the right/ command codes, table identifier or table values, subscript values and optional comments. The negative values are pointers.

4.4 Evaluation of Logical Expressions

Performing a statistical analysis, sometimes, data base should be divided into parts meeting requirements of the subsystem to be used. Decision rules dividing the data base are usually described by logical expressions of high complexity. In system SIS79 a generator procedure is applied to provide a simple method for defining these rules and to generate a program fragment performing the selection. /On mathematical logical investigations concerning this topic a lecture was given by I. Ratkó in Salgótarján, Hungary at the Conference on Mathematical Logics in Theory of Programming./

A	1	1	1	-	1	2	1	2	2	3	3	3	3	3
B	1	1	2	2	3	-	3	1	3	1	1	2	3	3
C	1	2	1	2	1	1	2	2	2	1	2	1	1	2
D=f(A,B,C)	1	2	2	3	4	5	2	1	4	1	6	6	4	6

Figure 2 Code table




```
$GRAPH
 $PARAM GRAPH='EXAMPLE',DATANA='C','A','B','B','A',
   NEWDAT='D',SACKNO=5,LEVELS=1,1,1,1,1,UPPBOU(1)=2,
 UPPBOU(2)=4*3$
```

Figure 3 Graph representation of function

```
$$$GRAPH  EXAMPLE
2    1          FIRST TABLE
1   -3 2,1
2    2        SECOND TABLE
    -4 1,3
     5 2
2    3         THIRD TABLE
    -5 1,3
     3 2
2    4               FOURTH TABLE
     1 1
    -5 2
     4 3
2    5         FIFTH TABLE
     2 1
    -4 2
     6 3
3
```

Figure 4 Commands to fill in tables

We performed interesting investigations in probability theory concerning the problem of selection [14] to find optimal strategies of file dividing.

4.5 Table-files, Output Tables

Results obtained by statistical data processing do not contain the data of individual items but those of typifying ones. Thus the files consisting of these raw data must be transformed into that of statistical data /frequency characteristics, code values' totals, quadratics sums, product sums, etc. /. Consequently, in statistical information systems it is not advisable to apply the languages developed particularly for handling and querying processes of raw data items. We achieved that after a suitable preprocessing /creating "table-files"/ a lot of different output tables can be obtained using a few seconds of CPU time /on HwB 66/60/ independently of the size of the sample [21]. It makes possible to perform statistical study of large samples in conversational processing.

5. REFERENCES

1. E. G. Codd, "A Relational Model of Data for Large Shared Data Banks", Comm. ACM, Vol. 13, 1970, pp. 377-387.

2. E. G. Coffman, P. J. Denning, Operating Systems Theory, Prentice-Hall, 1973.

3. M. Csukás, L. Greff, A. Krámli, M. Ruda, "An Approach to the Hospital Morbidity Data System Development in Hungary", Colloques IRIA, Tome 1, Informatique Médicale, IRIA, 1975, pp. 381-390. /paper presented at the Symposium on Medical Data Processing, Toulouse, 1975/

4. J. Demetrovics, "On the Equivalence of Candidate Keys with Sperner Systems", Acta Cybernetica, Vol. 4, No. 3, 1979, pp. 247-252.

5. D. E. Denning, P. J. Denning, M. D. Schwartz, "The Tracker: A Threat to Statistical Database Security", ACM Transactions on Database Systems, Vol. 4, No. 1, 1979, pp. 76-96.

6. W. J. Dixon, M. B. Brown /editors/, BMDP Biomedical Computer Programs /P-series/, University of California Press, Berkeley, Los Angeles, London, 1979.

7. M. Finkelstein, "A Compiler Optimization Technique", Computer Journal, Vol. 11, No. 1, 1968, pp. 22-25.

8. J. Foisseau, R. Jacquart, M. Lemaitre, M. Lemoine, J. C. Vignat, G. Zanon, "Program Development With or Without Coding", Software World, Vol. 12, No. 1, 1981, pp. 9-12.

9. L. Greff, A. Krámli, J. Soltész, "The Modeling of the Sampling Procedure for the Hungarian Hospital Morbidity Studies", Modeling Health Care Systems /editors E. Shingan, P. Aspden, P. Kitsul/, IIASA, Laxenburg, Austria, 1979, pp. 172-177.

lo. M. Hammer, G. Ruth, "Automating the Software Development Process", pp. 767-79o.

11. M. G. Kendall, A. Stuart, The Advanced Theory of Statistics, Vol. I-III, Griffin, London, 1958, 1961, 1966.

12. P. Kerékfy, "GENERA - A Program Generator System", Progress in Cybernetics and Systems Research, Vol. 11, Hemisphere, Washington, 198o. /paper presented at the Fifth European Meeting on Cybernetics and Systems Research, EMCSR'8o, Vienna, 198o/

13. P. Kerékfy, A. Krámli, M. Ruda, "SIS79/GENERA Statistical Information System", Progress in Cybernetics and Systems Research, Vol. 11, Hemisphere, Washington, 198o. /paper presented at the Fifth European Meeting on Cybernetics and Systems Research, EMCSR'8O, Vienna, 198o/

14. A. Krámli, M. Ruda, M. Csukás, M. Galambos, "Large Sample Size Statistical Information System for Honeywell Bull", Data Analysis and Informatics, /editor E. Diday/, North-Holland, 198o, pp. 457-462. /paper presented at the Second International Symposium on Data Analysis and Informatics, Versailles, 1979/

15. A. Krámli, P. Lukács, M, Ruda, "Probabilistic Approach to the Performance Evaluation of Computer Systems", Proceedings of the Third Hungarian Computer Science Conference, Vol. I, Invited papers, Budapest, 1981, pp. 51-64.

16. M. M. Lehman, "Programs, Life Cycles, and Laws of Software Evolution", Proceedings of the IEEE, Vol. 68, No. 9, 198o, pp. 1o6o-1o76.

17. N. H. Nie et al., SPSS Statistical Package for the Social Sciences /2nd edition/, Mc Graw-Hill, 1975.

18. J. Nievergelt, "On the Automatic Simplification of Computer Programs", Communications of ACM, Vol. 8, No. 6, 1965, pp. 366-37o.

19. B. Perron /editor/ et al., IDMS Concepts and Facilities, Cullinane Corporation, 1977.

2o. M. Ruda, "Some Estimates in Connection with the Critical Path Method", Project Planning by Network Analysis, Proceedings of the Second International Congress /editor H. J. M. Lombaers/, North-Holland, Amsterdam, 1969, pp. 2o7-215.

21. M. Ruda, "Statistical Information System with Health Service Application", MTA SZTAKI Tanulmányok, 87/1978, pp. 167-172. /paper presented at the Fourth Winterschool of Visegrád on the Theory of Operating Systems, Szentendre, Hungary, 1978/

22. M. D. Schwartz, D. E. Denning, P. J. Denning, "Linear Queries in Statistical Databases", ACM Transactions on Database Systems, Vol. 4, No. 2, 1979, pp. 156-167.

23. A. Wald, Sequential Analysis, Wiley, New York, 1947.

MTA Számitástechnikai és Automatizálási Kutató Intézete, Tanulmányok 133/1982 Proc. of RG-11, KNVVT

A RELATIONAL DBMS IN CONCURRENT PASCAL

I.Havel and P.Liebl
Matematicky ustav CSAV, II5 67 Praha I

Introduction

The aim of this paper is to show a possible connection between the fundamental ideas of parallel programming and those of the relational data base model. In particular, in the paper we describe how it is possible to implement, using the constructs and structures of the programming language Concurrent Pascal /CP/, a simple relational data base management system that functions in a multi-program environment.

The exposition is based on the assumption that the reader is acquainted with the basic notions of CP, as there are process type and monitor type, process variable and monitor variable /called here for short sometimes process and monitor respectively/. Our understanding of these notions differs from that of BH only in minor details.

The solutions presented here for the problem stated above was from the outset determined by certain facts and principles:

- the program is to run on a particular computer /the EC 8540/ which is characterized by a rather small fast memory /I28kbyte/, several disc units á 0,5MByte, varios I/O devices, and 4 interrupt levels
- the object program is to arise from the source program by cross-compilation on a standard EC I025 computer and should then be in use without alterations for a comparatively long time of several months or so
- no part of the source program, whatever small, should be written in an other language than CP; particulary not in assembler
- the program must contain as parts all problem oriented application procedures solving particular user requirements and written by user programmers as well as all system oriented procedures for the management of the data base and of all peripherals written by the system programmer of the user and by the

authors of the system.

Statement of the problem

Let us first briefly describe the requirements the project has to meet, partly given from the outset, partly derived by the authors from the expected mode of operation.

The task was to supply the EC 8540 computer with a software that supports the basic destination of that computer, which is to collect primary data at their source and to give them on in possibly modified, condensed or systemized form. Contacts with prospective users showed that a feedback relation to the source of the data is inevitable, and the situation is better described as data files operatively approached by queries and by update requirements. It appeared as natural to consider the data as a data base. The characteristics of the data given by this situation further suggested to apply to the data base the ideas of the relational model. This statement requires some clarification as to which characteristics we mean here and how they point towards the relational model. We shall touch some of these arguments when describing the declaration of the data base.

The special destination of the system made it possible to describe rather strictly the character of permitted queries and update operations: they will be routine operations whose exact structure, together with the solution of any possible result of the operation, will have been planned, approved, written down in Pascal, and processed by a compiler, long before the start of actual operation. There will be no evaluation of expressions in a query language, no unexpected demands.

Further consequences were derived for the utilization of storage. We decided that the data base should be stored on the disk units. On the other hand, all program parts, problem oriented as well as system routines, will permanently reside in the fast

core storage.

On the other hand, rather severe requirements are put to the system as to the parallel action of several programs, their mutual independence, servicing of queries and update requirements without undue delay, and security of the data from improper action, errors of the parametric user or his possible ill will. The hardware with its system of 4 interrupt levels has the potential to meet such needs adequately, and our task is to devise a software that exploits the capacities of the hardware. The proper answer appeared to be consequent programming in a high-level programming language. To express the mutual relationship of the individual program sections, the means of the language Concurrent Pascal appeared adequate. So, all the software describing the activities of the computer is considered as certain structures in one program. This program is written in CP (various parts of the source program come from different origins, some e.g. are copied from the manual), cross-compiled on a EC-computer and fed into the EC 8540.

Structure of the data base

Let us now describe the general structure of the DB, as it appears to the program parts that contain actions with the DB. The form of the declarations and statements defining the structure of the DB and the actions with it will be described in detail in further parts of this paper.

The DB consists of several relations in 1NF. Each relation has a name. An element of the relation can be considered as a line in the table that represents the relation. All lines of one relation have the same structure (1NF) described in the respective declaration. There is no limitation as to the number of attributes (except that it should be smaller than the number of bytes in the storage - limitations of this kind will not be mentioned further)

or to their type. Each attribute has a name and may be of any type permitted by Pascal - with the exclusion of the constructs file, pointer, and real, but with the additional possibility of subrange types with decimal limits (e.g. 1.01..2.00) which seemed to us particularly suitable with respect to the expected character of the data. One line in a particular relation is of the type record, where the field selectors are the names of the attributes of the relation. The number of elements of each relation (the number of lines in the table) is from the outset bounded from above by a number which is part of the declaration of the relation. Again, in the particular environment, the necessity to state beforehand a realistic upper bound to the expected number of elements of each relation should be felt as natural and will be satisfied easily.

The main simplification as compared with the standard theory of the relational model is that the unit of processing is one element of the relation, one line of the table, rather than the whole relation. This will be sufficient for the expected mode of operation, where the data come in, or are required in queries, one by one. Later we shall mention how the standard projection and join of relations can be programmed using the means at disposal.

Structure of the program

Let us now briefly describe the structure of the program, to be able to understand the position of the data base oriented statements within the program. As already said, all software describing the operation of the computer is one program in CP. The basic parts of that program are processes. A process is defined by a process type definition and becomes part of the program by the declaration of a process variable. Each process deals with one particular activity - an operation. An operation will typically

be the reaction of the computer to an event in the surrounding world as registered by one of the peripherals. It will consist of identification of the event, input of data (e.g. through action of a parametric user), verification of the data as to their formal, logical and as far as possible material correctness, one or several operations with the DB (a corresponding update, say), and possibly an output of a message. A quite similar formal structure is found in the reaction to a parametric query, as well as in other operations of this kind. The operation is represented by a program segment that has to start operation at a moment determined by outside events and ends after completing the above mentioned activities. Formally, the corresponding process has the program pattern of an endless loop that is halted at one or possibly several points by means of parallel programing (delay/continue in a monitor). The whole system of monitors and queues that governs the operations of I/O devices and connects them with the processes lies outside the scope of this paper and will not be mentioned further.

The overwhelming majority of the actions in an operation can be described by standard means of an ordinary programming language, particularly in Pascal. Outside this scope are matters concerning (a) peripherals and (b) the data base. We decided to treat these two groups as far as possiole uniformly. The second idea was to treat them as procedure calls. All statements having parallel programming character (i.e. the call of procedure entries in monitors) together with certain management are written out in procedures we call I/O procedures and DB procedures respectively. The DB procedures have a semi-standard form; principially they are written by the authors of the project but the user systems programmer has to fill in certain identifiers and constants.

The data sublanguage

Let us describe the DB procedures in some detail. They re-

present, from the user programmers view, the data sublanguage. As already mentioned, one unit of processing is always one line of a particular relation. In the relation, each line is characterized by its number. In general, the number of the line has no connection with its contents, the values of attributes of the element represented by that line. For each kind of action - search, read, write, add, delete - , for each relation it refers to, and for each process that is intended to perform that action, an individual procedure must be declared and incorporated into the process. The identifiers of these procedures are a local matter and are to be chosen by the system programmer; it is recommended however to construct them in such a way as to refer to the kind of action and to the name of the addressed relation. For the sake of exposition, assume for a while that the procedures shall work with the relation called employees. Quite generally, the passage of values between the process and the DB is exclusively effected through parameters (and not through e.g. global variables).

Take first the read procedure. Let it be called reademp. It has two parameters. The first parameter is of integer type and denotes the number of the line that has to be read. The second one is of record type, the type one line of the relation has, and denotes the variable into which the addressed line has to be transferred. So, the pair of statements i:=37; reademp(i,x) , where both i and x are variables declared in the process, has the effect of putting the variable x equal to the whole line no. 37 in the relation employees. To be able to work with the values of the individual attributes, it is sufficient to address now the fields like x.name (supposing "name" is one of the field selectors in the definition of the type of x, and at the same time the name of one of the attributes in employees).

A similar structure is found in the write procedure. The pair of statements i:=35; writeemp(i,x) replaces the values in

line 35 of the relation employees by the values that had at that moment been in the variable x. To perform an update that has to change only one field, to increase in line 12 the number of children by 1, say, the group of statements i:=12; reademp(i,x); x.children:=x.children + 1; writeemp(i,x) can be used: line 12 is placed into the variable x, one field is changed, and the updated line is stored back into the relation.

It is obvious that as long as the programmer has to work with the line numbers, the main ideas of the relational model cannot be utilized. In that sense a key role is played by the search procedure. It allows to single out lines according to their properties (in a quite general sense). The procedure searchemp has two parameters. The first is primarily the output parameter containing the number of the found line, while the second is the identifier of a so called condition procedure that is written by the programmer according to his needs. Let us demonstrate the situation by an example. Suppose we want to find in the relation employees those who work in department 7 and moreover are born before 1933. We have to declare a condition procedure (its identifier is local; let us choose it to be "ourman") with two parameters, the first of which is boolean. The declaration could be procedure ourman(p,x); begin p:=(x.dpt=7 and x.born<1933) end; the parameter x is, as seen, of the type of a line. When the statement searchemp(i,ourman) is executed in the program, the variable i contains as its value the number of the first line that contains data about an employee with the desired properties. So, the group of statements i:=Ø; searchemp(i,ourman); reademp(i,y); n:=y.name puts the variable n equal to the name of a suitable employee. If the relation contains no element of the desired properties, the output value of i is Ø, so that a more correct way of programming would be i:=Ø; searchemp(i,ourman); if i=Ø then actions to be taken in that case else reademp(i,y); n:=y.name .

The first parameter has also an input meaning: it denotes the number of the line after which the search is to start. So, i:=Ø means the search should start from i=1, the first line. This property of the first parameter can be used for finding several, or all, elements of the desired property. Suppose we have to fill the array n[1..1Ø] with the names of suitable employees (assume for example that we know there are at least ten of them). We write i:=Ø; for k:=1 to 1Ø do begin searchemp(i,ourman); reademp(i,y); n[k]:=y.name end . This example shows how the output value of i, denoting the number of the found line, can be used as input value for the next search that has to start from the next line.

The add and delete procedures require for their proper functioning the existence of an additional attribute in the relation that we shall call "valid" which is boolean and whose value false denotes this line belongs to a deleted element (or is empty from the beginning) and is prepared to be filled in by new values belonging to a new element. The procedure deleteemp(i) puts the value of "valid" in the i-th line equal to false. The procedure addemp(i,x) finds an invalid (empty) line and stores in it the value of variable x; the output value of i is the number of the just occupied line (and zero if there was no empty line). The value of "valid" in the newly occupied line is changed to true.

This sketchy description should show what kind of data sublanguage the user's programmer can use. One sees that although each statement works with one line (the search and add procedures naturally look through many lines in a loop, but they stop and offer a value as soon as they find the first line that suits them) and the line number appears explicitly, by properly choosing the structure of the data base and programming intelligently, one can actually work with a sequentially stored relation using the relational ideas. So, joins over two and even more relations can

be programmed elegantly by suitably declaring the condition procedure (the condition for the search in one relation then depends on values found in another relation).

In situations like that, the locking of relations is sometimes desirable. Otherwise, the element-wise mode of work would allow simultaneous approach to a relation by several processes. Declaring one queue and using delay/continue statements, the locking may be accomplished. It is then up to the system programmer to avoid deadlock.

The DB procedures themselves are rather straightforward procedures in Pascal including one or several calls of the write and read procedure entries in the respective relational monitor.

Data security

Of the many aspects of this complex category, some have been recognized as being important in the special conditions of the project and at the same time easily implemented.

The problem-oriented main bodies of the processes will often have to be written by non-professional programmers intimately acquainted only with the particular problem they deal with and only with the part of the DB connected with it. So, there must be some simple and safe way to shield the DB as a whole from mistakes in the processes. This safeguard is given by the necessity to explicitly state, by declaring and incorporating special DB procedures for each process separately, to which relations it has access and what kind of actions it may perform with them: whether only to read (one search and one read procedure), or also to update (an additional write procedure) or even to delete, say. We call this the principle of active security - not what is forbidden, but what is allowed, is to be stated.

Another powerful safeguard against local programming errors

that destroy non-locally data and program is given by the use of a higher programming language alone. Pascal, in a sense even more than Fortran, say, gives the possibility to detect bad programming and mistakes already at compilation time, and a further possibility to include into the object program powerful run-time checks. Although these slow down the execution of the program, they have been all included.

An important aspect of data security is the idea of the view. According to that idea, the individual programmer does not "see" the DB as it exists, but only a part (described in relational terminology by joins and projections). That prevents him and his process from changing or reading information which is not essential to the particular problem his process solves. The system we deal with here is concerned with data that are updated roughly as often as they are read. The general idea of the view however works satisfactorily only for reading.

With that in mind, measures derived from the view idea were reduced to two fields. One is the active security already descibed, which safeguards whole relations from write and/or read. The other is a device that makes accessible to each process only a chosen projection. This is accomplished by the way the individual DB procedures are written. The DB procedure communicates, as already stated, with the "deeper" structures of the parallel program by means of procedure entry calls. Data are transmitted through parameters of these calls. The unit of processing, namely the value of the parameter, is one complete line of the respective relation. The idea is now that the parameter through which the DB procedure communicates with the application program within the process may not contain the whole line out only a part of it. So, the program "sees" only certain attributes. That can be treated even so that the programmer never learns what the actual type of

a line of the particular relation is - he is informed only about the projection the DB procedures appear to him to work with.

Program parts defining the data base

Let us now say something about the program structures that represent the data base. To each relation there corresponds one monitor (one monitor type definition and one monitor variable declaration). This monitor - we call it a relational monitor - has two procedure entries, one for read and one for write. Each procedure entry has two parameters, one is the line number and one the contents of the line. The DB procedures in the processes work with these procedure entries in a rather obvious way. So, the read procedure simply calls the read procedure entry (as long as the data security measures are not applied; if they should operate, there are several assignment statements between the parameter coming from the relational monitor and the parameter going to the application program). The search procedure contains a simple loop that repeatedly calls the read procedure entry, and similarly the other DB procedures.

For reasons of time economy, each relational monitor contains a buffer for one line of the relation. That means that a variable of the type of a line of the respective relation is declared in the monitor (outside both procedure entries). The whole relation itself is not declared inside the relational monitor because that would mean that it is stored in the core memory. As it is, on the contrary, placed in the disk storage, the relational monitor, to recover one line from the relation or to store it, has to call, in his turn, on further structures in the program which might be characterized as logical and physical IOCS.

They are represented by one single monitor called DB, and further monitors that are wholly concerned with the management

of the disks. The DB monitor concentrates in itself the information about on which disk units and at which disk addresses the data base is stored, while each relational monitor contains the information where in the DB the respective relation is placed. From the view of the DB monitor, the data base is a one-dimensional array of characters (actually, for technical reasons, of integers), and the individual relations represent segments of that array. The transition from numbered lines which are of record type to the single characters is programmed in the relational monitors. The relational monitor calls the procedure entries in the DB monitor with parameters that describe the data to be transferred in terms of first address and length (besides the parameter that contains the transferred data themselves). There arise purely formal difficulties as the language Pascal, and its compiler as well, requires identity of type in assignment statements as well as in the parameters of procedure calls. They have been avoided by exploiting the fact that the compiler does not check the identity of types of parameters in calls that go from one monitor to another, as there is separate pre-compilation. With that reservation, the whole management of the DB (and similarly as far as possible of the I/O devices) is written out in consequent Concurrent Pascal.

The authors wish to express their gratitude to the members of Working Group 11 for valuable discussions on the subject.

[BH] Brinch Hansen P.: The programming language Concurrent Pascal. IEEE Transactions on Software Engineering 1, 2 (June 1975)

MTA Számitástechnikai és Automatizálási Kutató Intézete, Tanulmányok 133/1982 Proc. of RG-11, KNVVT

MODIFIABLE QUERY SYSTEM FOR CASUAL DATA BASE USER

Antonín ŘÍHA
Computing Center of Charles University,
Malostranské n. 25, 118 00 Praha 1

This paper reviews the work done and scheduled by the Computing Center of Charles University in Prague in the framework of the research task "System of contact with database in natural language". A query system called DOSYS / DOtazovací SYStém / or QASCU / Question-Answering System of Charles University / was developed by the team led by B. Miniberger in three years /1978-1980/. The description of DOSYS is adopted /with some modifications only / from /6/. The system is being further improved as a part of the research task "Mathematical problems of dialogue systems and knowledge representation" under the guidance of V. Koubek. A survey of future research tasks is adopted from /7/.

INTRODUCTION

For casual computer users, i.e. specialists in some non-technical field, nonprogrammers, no fully convenient means of communication with computer are available so far. Special artificial languages can serve this purpose to a certain extent only /2/. The necessary prerequisite for realizability of the so-called natural language communication with computer is to choose an application area the language of which is a naturally

restricted narrow subset of a natural language /2/. As experience shows, such areas do exist /5/. By naturalness of the restriction we mean that no other restrictions are imposed on a user than to keep within the problem domain. It is not, however, a sufficient condition for the effectivity of such communication. The best solution is to construct an information or query system as a whole - conceptual and internal schema as well as an interface.

SYSTEM OVERVIEW

DOSYS / Fig. 1 / is capable of answering queries asked as Czech sentences in written form. The first experimental problem domain of DOSYS is the enrollment procedure at all the Colleges of Charles University, Prague. The system is also intended to supply information concerning agricultural machine spare parts depots of a production company, and to supply information on parts of diagnoses of patients treated in one of the Prague hospitals. So far, DOSYS admits only complete mutually independent queries. It is not a dialogue system, in some cases, however, it informs the user that the query should be reworded or that there are some parts of the query that the system has failed to understand.

The natural language front-end of DOSYS consists of a main program QAS controlling all operations of the entire system and three subprograms - SLOV / dictionary retrieval/, LIAN /linguistic query analysis and the translation of the query into a formalized shape/ and SYNOD / reply synthesis /. The user communicates with QAS only.

The database component of DOSYS consists of a system RING and a program INTERPRET. The system RING is based on some ideas of the relational data model. The output of the LIAN program has a form similar to a narrow subset of the GET statements of ALPHA query language /1/. The program INTERPRET translates this output to the manipulation language of RING and performs some optimization, as well. In the actual version of DOSYS the feeding and updating of its database and its dictionary is separated from the system itself and is performed without any connection to natural language front-end. The first variant of DOSYS is working in the batch mode.

The input text /query/ is constituted by a sequence of word forms separated by spaces. Numbers and signs of punctuation are equally considered word forms. The analysis of a query is materialized in two steps. The first step consists in the dictionary retrieval performed by the SLOV program rewriting the query into the form of a string of words relevant for the reply of the system. To each relevant word a set of dictionary characteristics required for further linguistic analysis is attributed. To each word the following characteristics can be attributed: morphological characteristic / in most cases identical with the part of speech indicated in ordinary dictionaries/, semantic characteristic / the name of the semantic class to which the word belongs/, data characteristic / one or more conditions indicating the relationship of the word to the data stored in the database/ and characteristic of the anticipated context / the phenomena to be identified in verifying the correct comprehension of the query/. The dictionary implemented is of the form of an oriented tree with labeled nodes and edges. In the first variant of DOSYS the

dictionary is stored in the RING database.

The main principle of the DOSYS linguistic analysis is that of semantic condensation. It is based on the assumption that the factual piece of information in the text is borne by nouns. As far as syntactic aspects are concerned, no classical syntactic analysis is performed - only some structural relations between semantic units are covered. Morphological analysis is not performed at all. It is possible by means of semantic condensation to convert each query into a generalized semantic representation having more or less unified form and to neglect some formally expressed syntactic relations. In each query presented to the query system two parts can be distinguished - the object of the user's question and concrete data provided by the user / conditions /. The type of the query determines the operation to be performed / providing the list of items, the sum of items, calculating the percentage /. The linguistic analysis of the query constitutes the second step of query processing and is performed by the LIAN system of programs. The first program of the system marks some important parts of the query and checks the presence of some features, the second determines the type of the query and the object of the query. The third - sixth programs solve some ambiguities, the seventh processes the queries concerning percentage and the eigth processes the conditions /data/.

While the replies of DOSYS are elementary, they are sufficient and well comprehensible. The printed reply consists of three parts: the query itself, the reply and the list of word forms that the system failed to understand. The data retrieved from the database are fed into frames of answers prepared in advance.

The data of DOSYS are maintained in the RING database. It enables creating as many as 125 tables /relations/. The row /tuple/ lengths of relations are 30-900 bytes, the lengths of the keys being 20 bytes at most. The total volume of all data should not exceed 52 Mbytes. The data are stored on magnetic disc in a single physical file with a direct access method. The manipulation language of the RING system is materialized by three parameters of the CALL statement / type of the operation, relation and tuple specification, data/. It contains operators for defining and deleting relations and for reading, inserting and modifying individual tuples but, owing to its detailed nature, it is not appropriate for immediate wording of the LIAN output. Therefore, the program INTERPRET was designed to extend the functions of RING. There are two types of relations in the database: basic and derived. The basic relations contain new data, the function of the derived relations being similar to that of inverted files. The derived relations are updated automatically whenever some modifications in the basic relations are performed. In the first variant of DOSYS one basic and 38 derived relations are used, the total amount of data being about 20 Mbytes.

FUTURE RESEARCH AREAS

The experiment with DOSYS has demonstrated the realizability of natural language front-end for restricted problem area. At the same time, it revealed the limitations of the first variant. Therefore, we are now developing another query system KOMSYS /7/ / Fig. 2/. Similarly to IDA /9/, a user of KOMSYS is supposed to see the database in the form of a single relation with just the

attributes needed to satisfy his query. An internal partition of attributes to stored relations is known only to the system itself - each query is transformed automatically so that it accessed the relevant relations. Moreover, a user does not necessarily know the attributes stored. In his query he uses the so-called pseudo-attributes, corresponding to attributes in m:n way. From that it is clear that KOMSYS implements some ideas of 3-level database architecture according to ANSI/X3/SPARC proposal. Besides the queries, the system also performs some dictionary and/or database updates according to natural language commands. The linguistic query analysis is generalized and covers more types of queries. In cases of ambiguity, the program DIACON either resolves the ambiguity using the recorded information from the previous queries and answers or asks the user to specify the query. When the result of analysis is unambiguous, the system submits its interpretation of the query to the user for approval / in the form of natural language sentence/. On approval, the query is processed by the database component. The program DIVIDE transforms the query to the sequence of operations searching the corresponding relations in the database. The program CONQUER translates this sequence into the manipulation language of database system and checks the consistency. If the requirements of the user are not consistent with the contents of the database, a message is produced. The data retrieved from the database are first recorded by the DIACON for later use and then a natural language answer is constructed by SYNOD. The process of answer synthesis / or natural language query interpretation/ makes use of a dictionary, a frame grammar and a simplified morphological synthesis. DOSYS used two separate dictionaries for query analysis and synthesis, in KOMSYS

they will be united.

KOMSYS will be able to accept formalized queries as well. Analyzed by an appropriate analyzer, they will be transformed into the same intermediary language and processed in the same way as natural language queries. Natural language interpretation of the formalized query will be submitted to the user.

To create a query system is not an easy task. Moreover, the necessary restriction of the problem domain implies, in general, that the system constructed for one domain will not work for another / though also restricted / problem domain, and,to change the problem domain can be, in general, as difficult as to create a new system. Therefore, methods are studied how to perform at least a part of the task automatically.

E.g. the system LIFER /3/ generating natural language interfaces to existing software systems consists of two parts - the set of interactive language specification functions and a parser. Language specification functions are used to define an application language / a subset of a natural language/ that is appropriate for interaction with the existing software / DBMS /. LIFER application languages are specified by augmented context-free grammars. Each rule in the grammar includes a context-free production plus an arbitrarily complex response expression, which is the augmentation. The use of augmentation gives the LIFER parser the power of the Turing machine. The LIFER functions read the specifications interactively from the input and store them in the form of augmented transition trees / the simplification of augmented transition networks defined by Woods/. Using the specifications the LIFER parser interprets natural language sentences /queries/ and translates them into appropriate inter-

actions with application software. The application language can be extended by user by means of natural language commands.

We also intend to develop a metasystem that would generate the individual versions of KOMSYS for different problem domains. The description of a chosen problem domain written in a special formalized language will be on the input of the metasystem, giving it all the necessary information to create both natural language front-end and conceptual schema of the database. The creation of natural language front-end consists in the creation of a dictionary and modification /by substituting parameters mostly/ of all components / SLOV,LIAN etc./ so that they correspond to the problem domain. The final version of the metasystem should also design the partition of the database into individual relations and code data in the database and dictionary. It will also perform the modification of DIVIDE and CONQUER programs. An application of such metasystem will shorten substantially the transduction of our query system to a new problem domain and reduce the number of errors involved in that transduction. To create the metasystem, a sequence of theoretical and algorithmical problems should be solved; therefore, it represents a long-term task.

REFERENCES

1. CODD E.F., A data base sublanguage founded on the relational calculus. Proc.ACM-SIGFIDET Workshop, November 1971.
2. HARRIS L.R., Natural language and database query. Artificial Intelligence Corporation, 1980, 101-111.

3. HENDRIX G.G. et al., Developing a natural language interface to complex data. SRI AI Center, Tech. Note 152, August 1977.
4. MACHOVÁ S., MINIBERGER B., QASCU - Question-answering system for third-level computer users. Effective Computer Application, 1, 1981, 20-28.
5. MALHOTRA A., Knowledge based English language systems for management support: an analysis of requirements. Proc. 4th IJCAI, Tbilisi 1975.
6. BURÁŇOVÁ E., ECKHARDT P., GORALČÍKOVÁ A., CHALOUPEK P., KARNOLT J., KAŠPAR J., KOUBEK V., KRČMÁŘ M., MACHOVÁ S., MINIBERGER B., ŘÍHA A., ZÁMEK P., System of contact with data base in natural language. Technical Report VC UK - 1, Computing Center of Charles University, Prague 1980.
7. BURÁŇOVÁ E., ECKHARDT P., GORALČÍKOVÁ G., KOUBEK V., MACHOVÁ S., MINIBERGER B., ŘÍHA A., Komunikační systém KOMSYS. Report VC UK, Computing Center of Charles University, Prague 1981. / In Czech /.
8. ŘÍHA A., Creating natural language interface to data base. Proc. 4th International Seminar on Data Base Management Systems, held in Schwerin, December 1981, VEB LfA, Berlin, 1981, 198-206.
9. SAGALOWICZ D., IDA: An intelligent data access program. Proc. 3rd International Conference on Very Large Data Bases, Tokyo 1977, 293-302.
10. ECKHARDT P., Databankový systém RING. Studie VC UK-III-2-2/11-4, Praha 1979. / In Czech /.

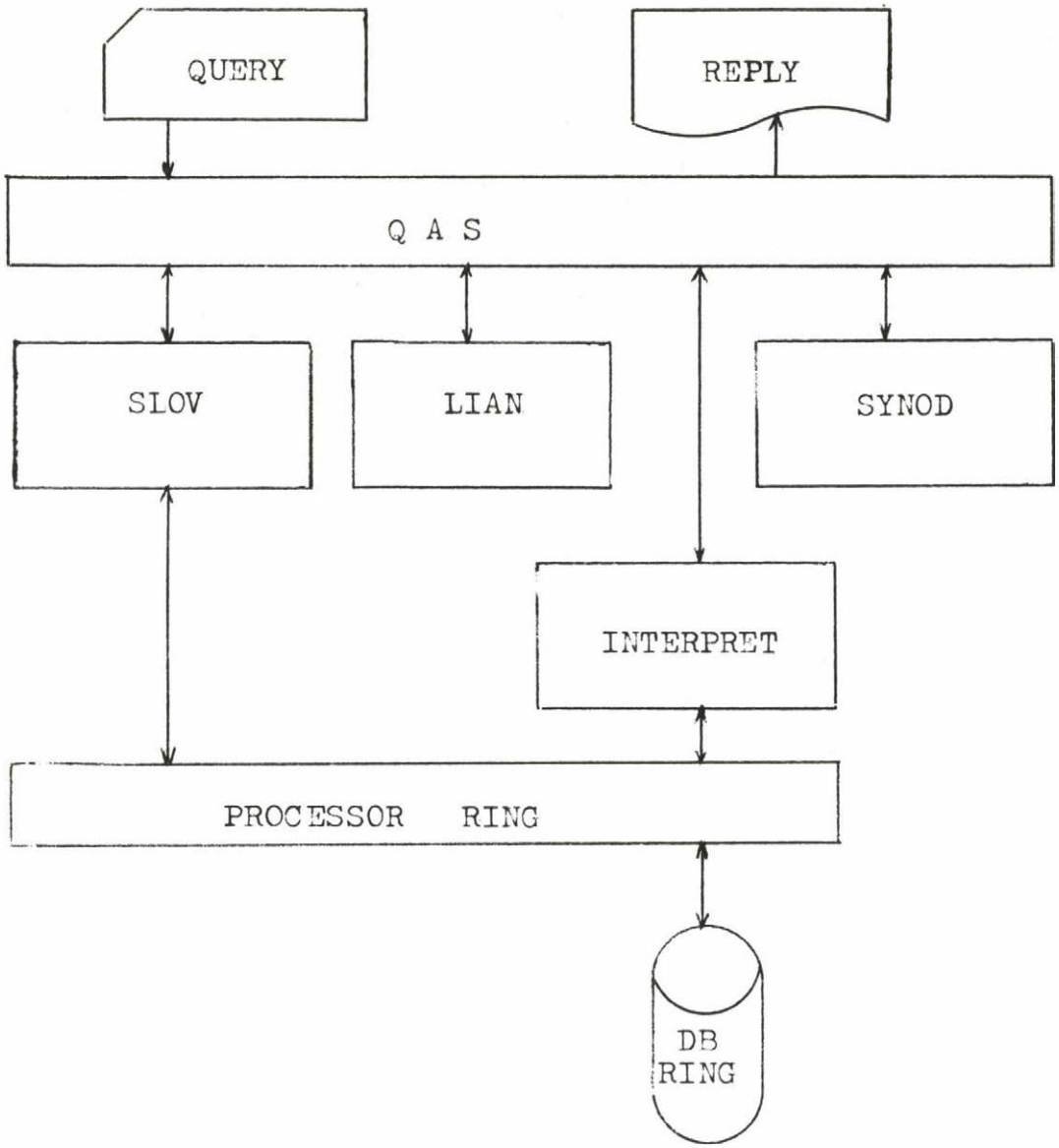


Fig. 1. Flow chart of DOSYS

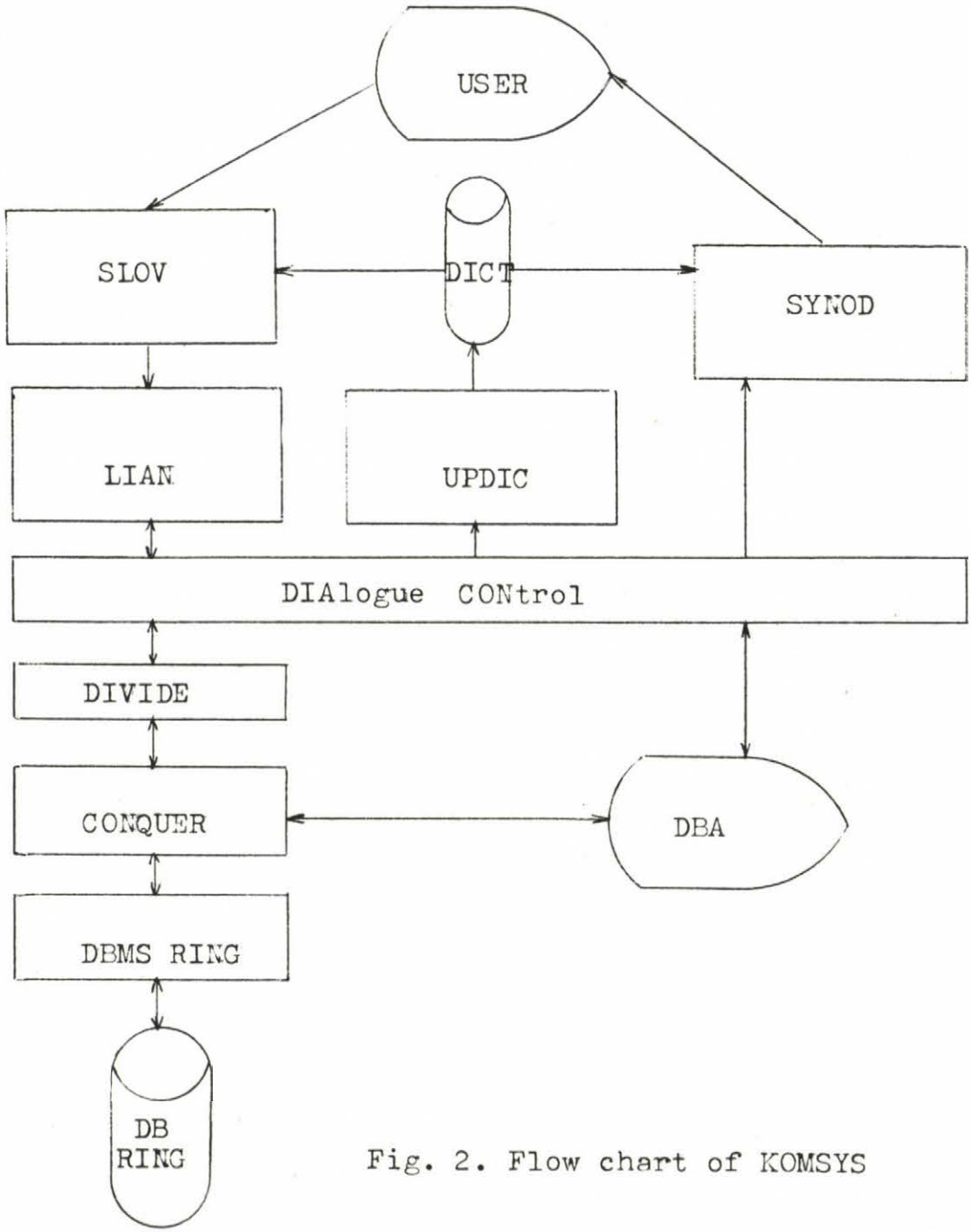


Fig. 2. Flow chart of KOMSYS

MTA Számitástechnikai és Automatizálási Kutató Intézete, Tanulmányok 133/1982 Proc. of RG–11, KNVVT

NATURAL LANGUAGE INTERFACES TO DATA BASES: A DAY-DREAM OR A REALISTIC GOAL?

by Willi Werner and Dietrich Koch

Zentralinstitut für Kybernetik und Informationsprozesse, Akademie der Wissenschaften der DDR

Abstract

In this paper we give an analysis of opinions and approaches to the topic mentioned above. The survey of existent systems and their approaches shows that the state of the art in natural language processing is capable of providing a sufficiently usable interface for certain "micro-world" applications. Yet, there are a lot of unsolved problems for further research. We recommend to overcome these gaps by a method which allows to concentrate on particular problems. Our proposal is to split the knowledge which is nessessary for the translation process into 3 parts:

- "back-ground knowledge"
- "data model knowledge"
- "DBMS knowledge"

Finally, some criteria for evaluating natural language processing systems are given.

I. Opinions

Natural language data base query has long been recognized as a useful application of artificial intelligence techniques. We restrict here to typed text legible by a machine. The essential difference between the "natural-language-like" query facilities and the true language approach is that the latter allows many valid phrasings of the same request, whereas the former only allows a single canonical wording, or perhaps a few very similar wordings.

There are a lot of different opinions about natural language interfaces (NLI), but nevertheless the efforts for this goal are ongoing. In the following we quote some of the remarks. It is clear that among the list of voices in favour of NLI the designers are included.

"We feel that the time is ripe for computers to be equipped for natural language systems which can be used by persons who are not trained in any special computer language." (WALTZ /1/)

"We have achieved a reasonably high degree of robustness, and this is largely due to the heavy emphazis on feedback to the user." (CODD about: experience with RENDEZVOUS Version 1 /2/)

The arguments against NLI can be summarized by:

"... it should be pointed out that no machine is currently as capable of handling natural language as a child of ten (indeed, there are some that claim (e. g. CHERNIAVSKY 1975) that there never will be)." (BROWN /3/)

"The quasi-natural dialog types are especially suitable for giving help to the user. Using it generally has the disadvantage of a lot of input typing which needs good typing skills and much time. Moreover the possibilities to understand natural language input are very limited and the implementation effort is rather

high; last not least this kind of dialog needs much core time. So very often the effort for quasi-natural language dialog is out of all proportion to the given advantages of this method." (DEHNING et al. /4/)

There are claims and proposals to use the NLI for creating a powerful conceptual schema of the data base. This enables the data base administrator to establish all complex relations between basic items in a given application-world.

"The role of natural language in such systems consists in the following: The natural language serves as base for the creation of the model of the outside world - the data base of the application field. In the communication process natural language may disappear - it is better to use a language which is near to the natural - a restricted "subject" language or a language of tabular type." (translation from VOSILJUS /5/)

II. The state

WALTZ called for papers to the topic NLI and received 52 articles /3/. In his paper the most advanced systems are mentioned, but no details about system features. We analysed a lot of these systems with respect to their natural language processing approaches. It is a difficult matter to compare and evaluate them, because the environments and the focus of research very often differ in many respects. It seems to be useful to distinguish between natural language interfaces to a given data base (NLI) and natural language understanding systems (NLU).

The focus of research for NLI is the question:

How is the mapping from the language-view to the external view in a multi-level architecture of a DBMS?

System	Type	Related person(s) and institution
PLANES	I	WALTZ, University of Illinois
LUNAR-ROCKS	I	WOODS, Bolt Beranek and Newman
SHRUDLU	U	WINOGRAD, SRI AI Center
LADDER, LIFER	I	HENDRIX, SRI AI Center
ROBOT	I	HARRIS, Darthmouth College, USA
RENDEZVOUS	I	CODD, IBM
DILOS	U	BRIABRIN, Acad. of Sciences, Moscow, U.S.S.R.
REQUEST=TQA	U	PLATH, Mathematical Sciences Dept., Yorktown
TORUS	I	MYLOPOULOS, Dep. of Comp. Science, Toronto, Canada
REL	U	THOMPSON, Univ. of California
PHLIQA1	U	SCHA, Philips, Netherlands
HAM-RPM	U	WAHLSTER, Univ. Hamburg, FRG
HANSA	U	WITTIG, Univ. Hamburg, FRG
...		BURGER, Syst. Devel. Corp., Colorado, USA
PLIDIS	U	WULZ, Inst. für deutsche Sprache, Mannheim, FRG
USL	I	LEHMANN, IBM Heidelberg, FRG
...	U	RIHA, A./SGALL, Charles Univ., Praha
FAS-80	U	KOCH/HELBIG, ZKI/ZFT-ROBOTRON, DDR

Table 1: Natural language processing systems
Type = I corresponds to NLI,
Type = U to NLU

NLU is more interested in recognizing, representing, and infering the knowledge which is contained in texts. Therefore, data bases in NLU need to be more complex. The used data model is more motivated from linguistic and logical points of view than from efficient as in commercial DB. NLU is up to now in an experimental state. Pilot systems for demonstration NLU-capabilities are knowns as "Question-Answering systems". After a listing of advanced language processing systems, known to us, we give some criteria for evaluating their power.

III. Basic approaches

In the multi-level architecture of a DBMS we have different levels for the view of the applied data model. For each level exists a certain class of users who use this level as interface. The related languages may be taken from the language categories given in Table 2. This does not imply that the language level for the conceptual schema is any kind of standard programming languages; it may be that the DBMS has the facility to manage the conceptual schema by natural language or any kind of frame driven language.

In Fig. 1 we have outlined a basic approach for language translation. On the left side we have the case that

either the source expression and the goal expression can be proved to be equivalent by simple searching; a trivial case where nothing has to be translated

or the most complex case, where all steps necessary for the translation, as can be seen on the right side, are incorporated in a single step. (nothing can be seen).

On the right side we have all steps, necessary for a powerful

translation, separated according to their task. It is easy to see that this method allows better to concentrate on particular problems.

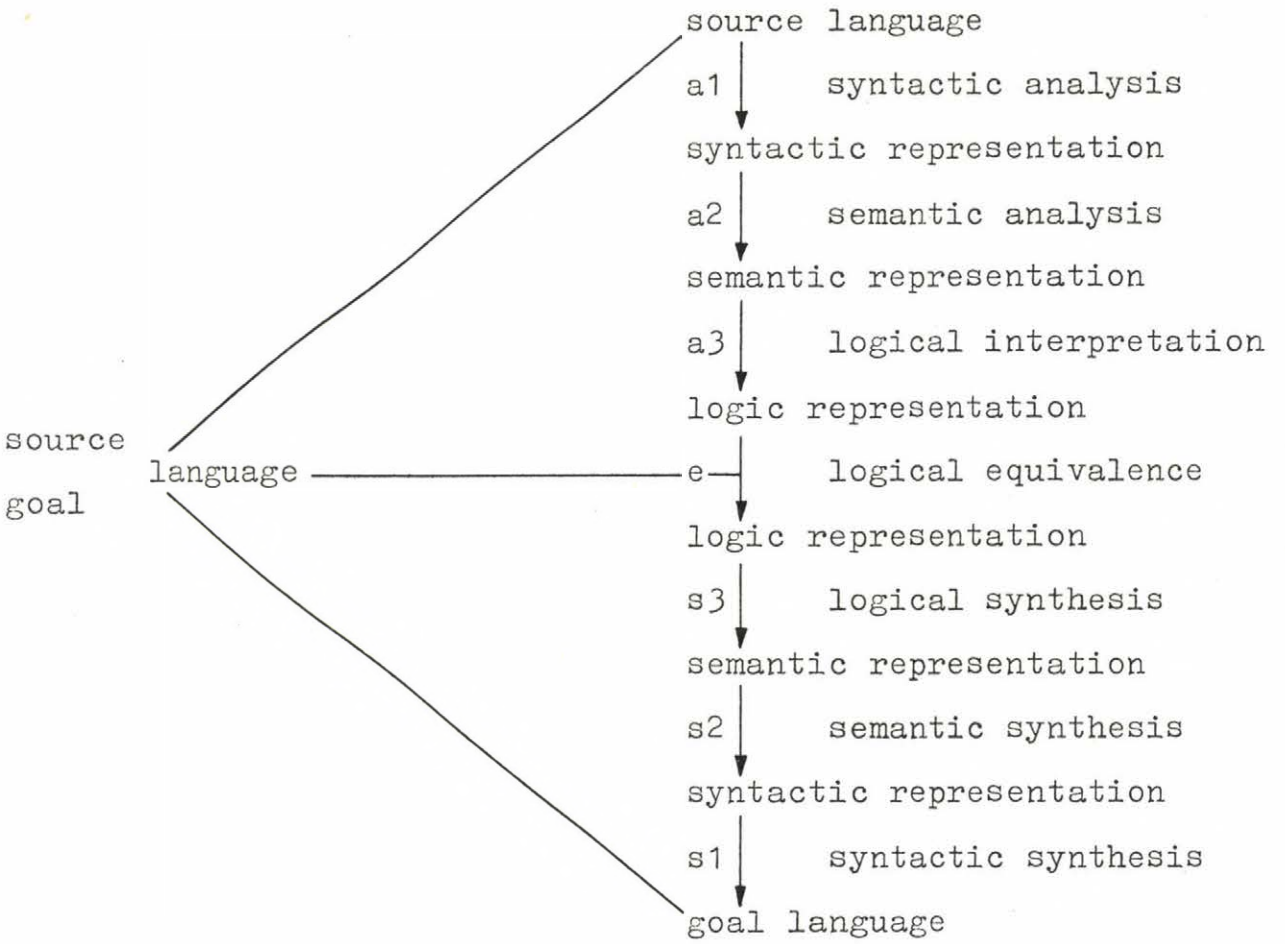


Figure 1: The basic scheme for translation one language to another

Level	Basic units/ (theoretical base)	Examples (incomplete)
NL	text, sentence, phrase (dependence, categorial grammars)	Engl., Russ., Germ.
High-level command language (HCL)	frame, graph, network (graph theory)	KRL, LIFER, ATN, PLANNER, SDLA
	predicate-calculus-expr. (predicate calculus)	PROLOG, DEDUCE, KS, PCF-2
	relation, tuple (relational calculus)	ALPHA
	relation, tuple (relational algebra)	SEQUEL-2, ISBL
	set (set algebra)	SETL
High-level programming language (HPL)	set, tuple, tree, list, atom, string, number	QA-4, APL, SAIL, LISP, PL/I
Standard programming language	tree, array, strings, number, adress	FORTRAN, ALGOL, COBOL, ASSEMBLER
Machine language	adress	

Table 2: Language categories

In order to prove that two semantic structures are equivalent, we need a logical interpretation for each of both what may result in an explicit logical representation.
In Fig. 1 nothing is said about the effort for each single step above and below the equivalence level. At first glance the process seems to be symmetrical, but this is not the case. Generally, we can say the translation effort increases with the distance between the language-levels of the source - and goal language, their absolute distance from the machine language, and of course with the complexity of language phenomens which can be handled. The effort for synthesis is generally minor than for analysis. In order to illustrate these (trivial) statements, we consider some translation cases, where the a_i, e, and s_i name the related steps.

1. NL —paraphrasing→ NL

   (a1, a2, a3, e) expensive
   (s3, s2, s1) less expensive

   This is why s3, s2, and s1 can operate on recognized structures, with few rules, and possibly canonical syntactic output.

2. NL —translation→ NL

   a) Russian → English

   (a1, a2, a3, e) expensive
   (s1, s2, s3) less expensive

   b) English → Russian

   (a2, a3, e, s1) expensive
   (a1, s2, s3) less expensive

   This is why syntax in Russian is more difficult and more

powerful than in English.

3. HCL —translation→ NL

for example

ALPHA ⟶ German

(a1, a2, a3, s1, s2, s3) little expensive
e expensive,

because the production rules for syntax, semantic and logical interpretation of ALPHA are small and the effort for synthesis remains also small, after a equivalent "German view" is found.

Methods for transforming linguistic structures into a formal query

We analysed a row of NLI's and found that their applied methods can be subdivided into 5 classes:

I. pattern matching
II. direct translation
III. translation of syntactic constituent lists
IV. translation of syntactic trees
V. translation of semantic structures

The difference between these classes consists in the degree of their distinction between linguistic knowledge and "data model dependent knowledge".

pattern matching

Every possible question form is associated with a corresponding formal query pattern. The problem is solved after that pattern is found and the variables of them are substituted by their natural language constants. This method is for aims of translation out of

interest.

direct translation

After the paraphrasing the original question into a canonical form "data model frames" are used for construction of the formal query. The variables of the "data model frame" are complex and can be fitted by the results of applied subnetworks. This method is used in RENDEZVOUS, PLANES /1/ and LADDER /5/.

translation of syntactic constituent lists

Syntactic parsing the sentence yields a list of constituents. The dependence tree for formal semantic representation results from "data model" given dependencies. This method was applied in PLIDIS /6/.

translation from syntactic trees

The dependencies of the constituents of a sentence are recognized by linguistic motivated knowledge and a syntax tree is obtained. This tree is by domain dependent rules associated with the corresponding formal language expression. This method was used in LUNAR by WOODS /7/.

translation of semantic structures

After obtaining a semantic representation by appling linguistic motivated case frames this structure will be transformed into its "data model", "DBMS-dependent" and "back ground knowledge" equivalent. This method seems to be very expensive but it guarantees a very high degree of knowledge separation. It is applied in PHLIQA1 /9/ and proposed by us.

The process of the semantic translation from the linguistic

semantic representation to the semantic representation in a given data model in a DBS is very difficult and the crucial point for the NLI.
We proposed to split this process in some subtasks in order to differentiate between "background knowledge", "data base knowledge", and "DBMS built-in knowledge" (WERNER, /8/). Similar argumentation can be found in (SCHA, /9/):

He wrote:
"--- Prior to the design of the system, we made a careful analysis of what is involved in the question answering process. This led to the idea of multilevel semantics: the distinction of three different levels of analysis which the question passes before it leads to to an answer. At each of these levels, an expression of a formal logical language is used to represent the meaning of the question. At every level, a different level language is used. The three languages have the same syntactic and semantic structures. They differ in the constants they contain. These constants represent different kinds of entities in each case:

- the constants of the English-oriented Formal Language (EFL) represent the unanalyzed meanings of English words and grammatical functions.
- the constants of the World Model Language (WML) represent the concepts that constitute the subject domain of the system.
- the constants of the Data Base Language (DBL) represent the "record-types", "data-items", and sets which constitute the data base, and the available logical and arithmetical procedures."

In order to overcome the crucial gap between the linguistic and data base model, the user and the system must cowork together in a

clarification dialog. The user has interactively to control and to guide the system's way of interpretation. The system has to provide a feedback to the user and to give him some "menu"'s for help.
An experimental approach for this method is given in Figure 2 (CODD /10/, where you can find a good impression about RENDEZVOUS capability).

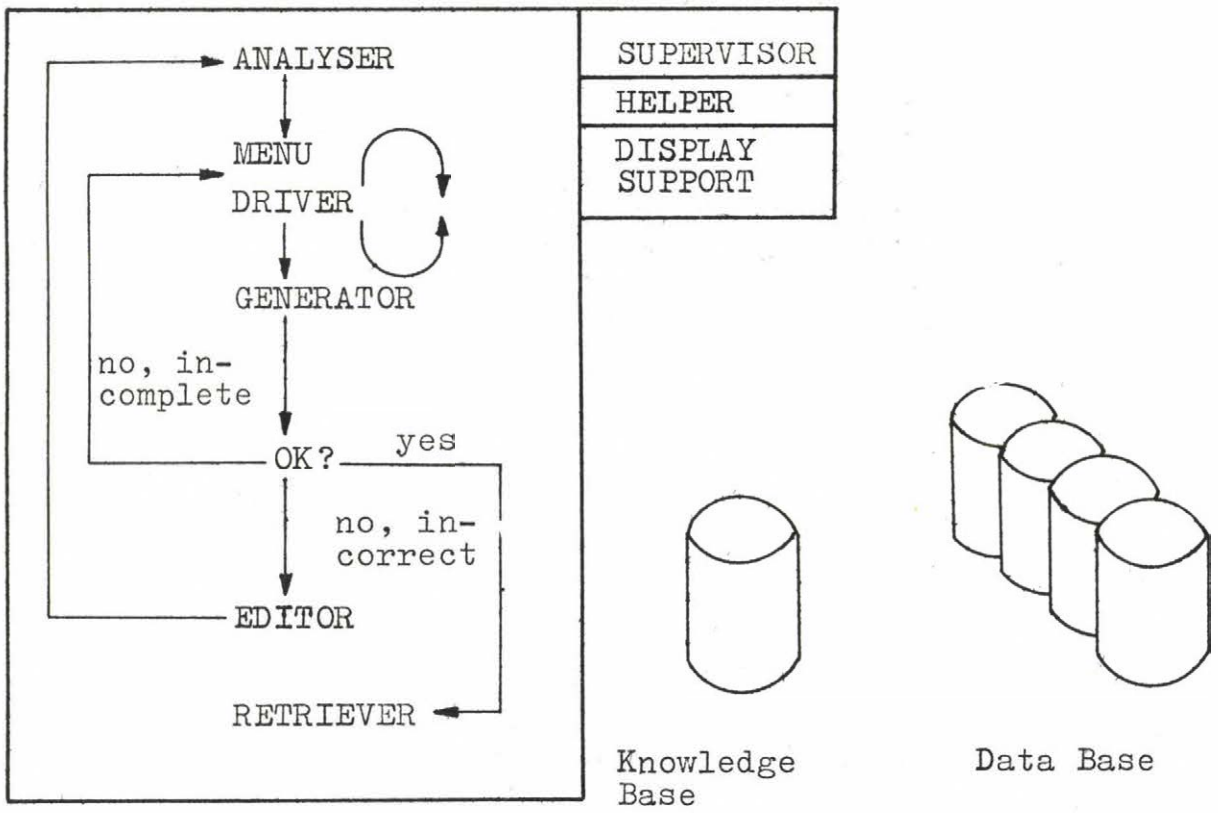
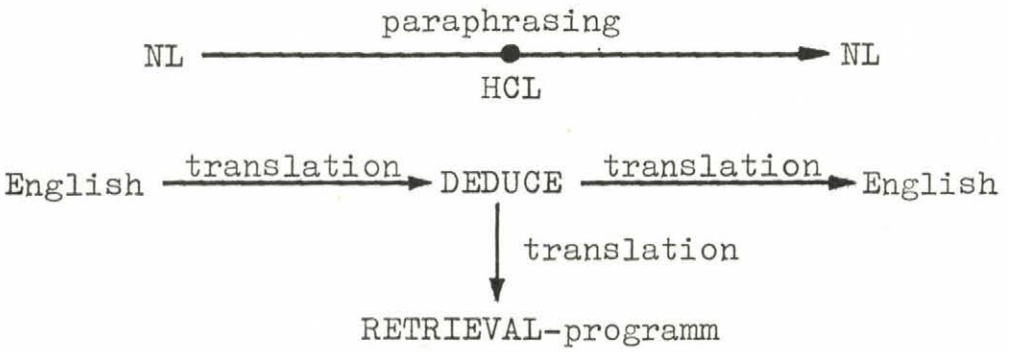


Figure 2: RENDEZVOUS Version 1 system

The ANALYSER translates the original question into the formal

query language DEDUCE.
The MENU-DRIVER tests its logical completeness and poses only questions to the user that will yield a logically complete formal query.
The GENERATOR retranslates the formal query in English.
We can therefore summarize the entire process in the following scheme.



IV. Measurements of capability and progress

Before anybody judges about the quality of a given NLI he should try to fill out the following checklist:

A. linguistic capability

ability to solve:

a) sentence references?
b) pronominal references?
c) coordinations?

ability to use:

a) ellipsis?

b) adverbial modification?
c) adjectival modification?
d) relative clause modification?

state:

a) which phenomena can be handled?
b) which phenomena can not be handled?

B. logical capability

which kind of inference rules are incorporated?
how quantification is solved?
how negation is solved?

C. convenience and perspicuity

interactive possibilities:

a) extendable?
b) model correction?
c) spelling correction?

feed back:

a) self explainable?
b) menu support?

D. portability

to other machines?
to other subject domaines?
to other applications?
of single modules?

E. efficiency

An overall measure for this is the time necessary for the correct translation of NL into the used formal query language. But, you

must be careful. It depends on:

- speed and resources of the used computers,
  what they are:
- the organization overhead,
  what management system is used?
- the chosen transforming language,
  name it:
- the mode of working,
  interpretative?
  compiling mode?
- the level of the target language,
  name the target language:
- the quality of particular algorithms,
  name them and give a measure for their quality:
- the cleverness of their implementation,
  give measures for such cleverness:

F. What about the users experience?

Consult them!

V. Conclusion

With good coordinated research and development the goal NLI can be a realistic one, but today only for restricted languages and restricted "subject domains". The users of a modern DBMS need it.

/1/ Waltz, D. L., The PLANES System: Natural Language Access to a Large Data Base. Report T-34, 1976, Univ. of Illinois USA

/2/ Codd, E. F., RENDEZVOUS Version-1: An Experimental English-Language Query Formulation System for Casual Users of Relational Data Bases. IBM, Research Report, 1978

/3/ Waltz, D. L., Natural Language Interfaces. Sigart Newsletter, No. 61, 1977

/4/ Dehning, W. et al., The Adaption of Virtual Man-Computer Interfaces to User Requirements in Dialogs. Springer, 1981

/5/ Hendrix, G. G, et al., Developing a Natural Language Interface to Complex Data. ACM Transact. on Database Systems, vol. 3, No. 2, June 1978

/6/ Berry-Rogehe, G. L., Wulz, H., The Design of PLIDIS, A Problem Solving Information System with German as Query Language. Mannheim, 1976

/7/ Woods, W. A. et al., The Lunar Sciences Natural Language Information System. BBN Report No. 2378, 1972

/8/ Werner, W., Ein Ansatz für ein NLI zu relationalen Datenbanken, Schriftenreihe der TU Dresden, Heft 44/80

/9/ Scha, R. J., Philips Question-Answering System PHLIQA1. Eindhoven, Netherlands, see /3/

/10/ Schneiderman, B., Data Bases: Improving Usability and Responsives. Academic Press, 1978, p. 3 - p. 27

/11/ Vosiljus, S. K. i drogich, Voprosy realizacii reljacionogo interfejsa
PVK Banki dannych, Tbilisi 1980, Sekcija 2

A TANULMÁNYSOROZATBAN 1981-BEN MEGJELENTEK:

116/1981 Siegler András: Egy 6 szabadságfokú antropomorf manipulátor kinematikája számítógépes vezérlése

117/1981 Knuth Előd – Radó Péter: Principles of Computer Aided System Description

118/1981 Demetrovics János – Gyepesi György: Általános függőségek és lekérdezéssel kapcsolatos algoritmusok relációs adatmodellekben

119/1981 Sztanó Tamás: REAL-TIME programrendszerek eseményvezérelt szervezése

120/1981 Szentgyörgyi Zsuzsa: A számítástechnika műszaki fejlődése és társadalmi hatásai

121/1981 Vicsek Tamásné (Strehó Mária) : Vizsgálatok a kezdeti érték problémák numerikus megoldásával kapcsolatban

122/1981 Andó Györgyi – Lipcsey Zsolt: Sztochasztikus Ljapunov módszerek és alkalmazásaik

123/1981 Márkusz Zsuzsanna: Intelligens interaktiv rendszerek elvi problémái

124/1981 Márkusz Zsuzsanna: Logikai alapú programozási módszerek és alkalmazásaik számítógéppel segített építészeti tervezési feladatok megoldásához

125/1981 Fabók Julianna: Software implementációs nyelvek

126/1981 Várszegi Sándor: Multimikroszámítógépes-rendszerek

127/1981 Lipcsey Zsolt: N-személyes minőségi differenciáljátékok késleltetéssel és késleltetés nélkül

128/1981 Böszörményi László: Multa-task rendszerek fejlesztése magasszintű nyelven

129/1981 Tóth János: A formális reakciókinetika globális determinisztikus és sztochasztikus modelljéről és nöhány alkalmazásáról