tanulmányok 194/1986

MTA Számitástechnikai és Automatizálási Kutató Intézet Budapest

Magyar Tudományos Akadémia Számítástechnikai és Automatizálási Kutató Intézete
Computer and Automation Institute, Hungarian Academy of Sciences
Исследовательский институт вычислительной техники и автоматизации
Венгерской Академии наук

СТАТЬИ КОНФЕРЕНЦИИ КНВВТ ПО АВТОМАТИЗАЦИИ ИНФОРМАЦИОННЫХ ПРОЦЕССОВ НА ПЕРСОНАЛЬНЫХ ЭВМ

Будапешт, 5-9 мая 1986 г.

Том II.

KNVVT CONFERENCE ON AUTOMATION OF INFORMATION PROCESSING ON PERSONAL COMPUTERS

Budapest, May 5-9, 1986

Vol II.

Будапешт, 1986

Budapest, 1986

Tanulmányok 194/1986
Studies 194/1986

A kiadásért felelős:

DR. KEVICZKY LÁSZLÓ

Редактор:

ИШТВАН РАТКО

Editor:

ISTVÁN RATKÓ

ISBN 963 311 223 0
ISSN 0324-2951

87/65. Alfaprint
F.v.: Barabás Gábor

CONTENTS

Page

ABOUT THE DOCUMENT STRUCTURE IN OFFICE INFORMATION SYSTEMS

P.K.Azalov F.I.Zlatarova

ABSTRACT In this paper we focus on the document organization and the management aspects of office information systems. General problems about document models and data types used in their definition are considered. A concrete office document model called Object Document Model (ODM) is described. The object is a program unit representing a document type such that applying some determined rules, different document instances are generated. In the data base field there are used three basic notions: data model, schema and data base. Similarly here are considered the next three notions: ODM, object type and document instances base.

1. INTRODUCTION

There is a growing interest among computer science researchers about information systems that handle complex data such as text, attribute data types (integers, reals etc.) and image. Now these systems are changed to taking into account new aspects. A short review of the basic possibilities of the data storage and the data retrieval computer systems suggest that their evolution is closely connected with the complicating of the data handled. The main evolution stages with respect to the object structure presented in these systems are:

a) informational objects having fixed structure (fixed number of fields, fixed length of each field);

b) informational objects having variable number of fields and variable length of some of these fields;

c) informational objects being a form;

d) informational objects containing nonstandard types, including graphics elements;

e) informational objects having semi-free format (see 2.2).

Obviously, the objects corresponding to e) are the most complex ones and they contain in themselves the objects from the other types. It can be considered that those informational objects not being especially mentioned in the above classification may be situated between the objects of type a) and e) with respect to their structure complexity and their building components. It is very interesting to consider the problem about the existence of a general system having possibilities for managing informational objects from the simplest type a) to the most complex type e). Systems which need these possibilities to manage data with so large range of the permitted structure are the office information systems. Some of the functions that these systems may provide are creation and filing of office information, content addressibility of office documents, automatic insertion of documents in a paper from and document transmition and reconstruction in a different site. Sometimes the realization of office information systems may be performed by means of DBMS.

2. DOCUMENT MODEL

2.1. What is a Document Model?

The basic information carrying entity in office information systems is the document. Documents are used to communicate information in office. A document can be a data base record, a text document , an image or any combination of the above [2]. There exists a similarity between the document model in a document information system and the data model in a DBMS [1]. The three concepts characterising a DBMS are: a data model, a schema and a data base. They are to be found in office information systems too. For example, in a form system such as OFS [4] the corresponding concepts are: form description language, form types and form instances. The type concept described in this system is definitely useful when dealing with forms, but this is not always the case with more general documents. In general, the document model (DM) may be determined by a set of generating rules (R), with respect to which the documents are built [3]. But the structure specifications do not assure possibilities for the complete interpretation of the document semantics and of their application mode. It is realized by specification of the document operations. The set of generating rules R for building of documents is usually called a data definition language. A DM must support types with a high degree of flexibility in their structure and provide as much knowledge as possible about the structure of a given office document in order to assist in its creation, storage and retrieval.

2.2. Data Types Used in Document Models

Building elements of each office document are the primitive types (or basic data types), such as an integer, a real, a string, a boolean and a pointer. It is not difficult to give examples where the first four types are used. The type pointer may be used by refering to a document from the same or an other type, or by refering to a paper document, a table or a graphic object. Using pointers of the first type the possibility to link documents in a correspondence is obtained. The other pointers are very useful by the creation of a document dossier, of documents having appendices and s.o. With the help of the primitive types more complex nonprimitive types are defined such as a date, an address, a telephone number and others. The definition of these types is performed in two levels: external and internal representation. Usually for the internal representation of the type date an ordered triad of integers is used, corresponding respectively to the day, month and year. This representation must allow cases of incompleteness and indefiniteness for the values of some components. Often the external representation is like a string having variable structure and lenght. For example:

May 15, 1985
15.05.1985
15 of the last month
New Year

Questions like these appear defining and analysing some other composite types.

There exist two basic kinds of documents used in the office practice: formatted documents (F_document) and semi-free formatted documents (S_document). It is possible to write:

```
<Office_document  ::= <F_document>|<S_document>
```

The wide-used formatted office documents often are defined as forms.

Obviously, the formatted documents are very near to the managed in DBMS informational objects. They may be considered like a sequence of formatted elements (f_elem).

```
<F_document> ::= <f_elem>|<F_document> <f_elem>

<f_elem> ::= <f_field> | <standard_part>|
             <standard_part>  <f_field>|
             <f_field>  <standard_part>

<standard_part> ::= <string>

<f_field> ::= <prim_elem> |<Nprim_elem>

<prim_elem> ::= <integer> | <real> |<string>|
                <boolean> | <pointer>

<Nprim_elem> ::= <tel_number>| <date>| <address>
```

Each formatted element may be a formatted field (f_field), may be a standard part (standard_part) which is nothing more than a string or both in the same time.

The letter is a typical example of a semi-free formatted document. Usually the office letters have some obvious elements. In fact in the office practice there are used letters having arbitrary structure (T_object). The free text and the graphics objects (G_object) are typical elements for semi-free formatted documents.

```
<S_document> ::= <s_field> | <S_document> <s_field> |
                 <S_document> <f_field>
<s_field> ::= <T_object> | <G_object> | <table>
<T_object> ::= <string>
<G_object> ::= <graph> | <pie_chart> | <histogram>
<table> ::= <table_row> | <table> <table_row>
<table_row> ::=<f_field> | <table_row> <f_field>
```

There exists an essential question, which must be considered. It consists in determing the difference between the free format and the string in the case when the free text is represented by a string. The answer is the following: both types differ only in their semantic. The definition domain of the field having the type string is well known beforehand. This fact permits the analysis and the management on these fields. An example of such a field is the attribute EDUCATION in the personnel card of every employee working in an enterprise. The semantic content of the fields having the type

T_object is not known, thisway the management of such fields can not be established in advance. It is possible to accept this type to be "undetermined" or "unknown". Sometimes we can not type a field of a document. That is, we do not know a priori that all the instances of that field are of a given type, or even for that matter that it is one data type. Take a letter as an example: if the body of the letter is a field, then any particular instance may have one or more data types as the content of that field.

3. THE OBJECT DOCUMENT MODEL (ODM)

3.1. A Document Description Language

Up to here considered were the building elements of the office documents. Document creation and management could be made using the experience from the work with DB. Formal language facilities for description of the objects corresponding to a given object area exist for each data model. The availability of language tools for object description for an office activity, i.e. the office documents, will permit the definition of document types called further on object types taking into account some more general functions. The real documents used in a concrete office will be the document instances generated from each document type, existing for this office.

3.1.1. What is an object type?

The object type is a program unit in the commonly accepted sence, whose components can be descriptions of: variables, files, rules for value computations, object type performance conditions and document instance building operations, Differing from the usual program modules, the object types can be main or subordinate at the same time. The general structure of the object types is the following:

```
OBJECT <name> ;
VAR <list of variable descriptions> ;
FILE <list of file descriptions> ;
FUNCT <rule descriptions> ;
COND <conditions descriptions> ;
MODULE <list of instructions> ;
end.
```

In conformity with his structure the object type has some similarity with the PASCAL-like program units. But the similarity is only apparent. The differences consist in:

- the type and the mode of definition and the utilization mode of variables, files and functions;
- the type and the mode of operations performance building the module;
- the activating mode of the object type;
- the results from the activation of the object type.

It can be admitted that the object type represents a specification of the attributes common to the document instances, generated by this object type. In other words, the object type (described by the document description language) represents exactly what is a schema (described by the data definition language) for the data base.

Variables

There are used variables having primitive and nonprimitive type. The first ones are given by FORTRAN-like specifications: I, F, A, L. There are permitted also the types: a date, an object type, a graphic object, a procedure and a metatype, all they noted respectively by D, O, G, P, T. The definition of all variables is made in the way, shown in the next example:

```
VAR X:F7.2, A:G, ALPHA:A7, BETA:A15 ;
```

Files

The notion file is used in his known sence. Each file is determined by its name and an ordered list containing field names, each of them defined like the usual variables.

```
FILE  F1(NAME:A10, NR:I5, S:F3.1)
      F2(NM:I4, DATE:D, ABST:A240) ;
```

To use files in object type definition is not obligatory, but in most cases it helps the automatic synthesis of document instances. When the documents are formatted, they can be elements of such files in a DB and so they can be useful by the creation of new office documents (not necessarily formatted).

Expressions

The expressions are used for value computation. Permitted expression types are: I, F, A, L and D. They are simple or conditional. The values, participating in expressions are constants, variables and file fields. The expression values are assigned to variables with the help of the assigment instruction.

Examples:

```
1) F: I5=A+7
2) Y: A1=if L=0 then 'M' else 'W';
3) NAME: A16=CITY+'CITY'-_'+CODE;
4) R: F5.2=if A=2 then 'X+Y±' + STR(ST)
                   else PROG;
```

The examples 1) and 3) illustrate simple expressions from the types I and A. The examples 2) and 4) represent conditional expressions. In the last example in the case of A = 2 the procedure PROG will be activated.

Conditions

An object type is in an active state if an appeal from an other object type or directly from the office information system is manifested to it. The object instance generation of a given object type is possible only if the respective object type is in an active state and if the conditions in the section COND are fulfilled. These conditions are written like logical expressions,

Instructions

The document instances are generated due to a sequence of instructions, contained in the section MODULE of the object type. There exist two kinds of instructions:

a) reference to: variable, file, file field, expression, procedure, object type, editing function, DB-function;

b) control instruction: unconditional, conditional and cyclic.

The aggregate of document instances, generated by the object types described in ODDL will be called an object data base (ODB). The presence of files used as data structures in the object types building and also the generating of concrete document instances creates a direct link between data in DB and documents in ODB.

3.2. Document Operations

The document operation is a basic operation. It can be considered like a composite operation of the following two operations: object document creation and document instance creation. The first is performed with the help of a specialized editor, whereas the second is performed automatically using previously created object type and introduced in the DB data.

The retrieval of a document in ODB is a very important and complicated operation in the document management. The semi-free format of the documents in ODB and namely the presence of heterogeneous building elements (text, graphics data) submits very serious problems about their physical organization and the method for their retrieval. The document instance visualization must permit the output of mixed data types (text and graphics data) on the screen or on paper. On the basis of existing references between documents (a document type, which has a reference to another document type, has also references to the document instances generated by the second document type) it is possible to define the operations union and correspondence. Due to the first of them, instances of different types can be arranged in groups, and due to the second one it is possible to group instances of the same type. The operation document modification is not so typical, but sometimes it may be used. Document management for documents of a given type may include the performance of some arithmetical operations on numerical elements in a table, described according to this type, also the sorting of documents in respect to concrete criteria and s.o.

4. CONCLUSION

Management of unformatted data presents a variety of new possibilities and perspectives for data base management researchers. Here considered was only a part of the problems in respect to the office documents structure. Very interesting but quite sophisticated are the corresponding questions about software architecture of office information systems, physical document base design techniques, image processing techniques, concurency control, security, version support etc.

REFERENCES

1. F.Rabitti, A Model for Multimedia Documents, Office Automat'on (edited by D.Tsichritzis), pp.227-250, Springer-Verlag, 1985.

2. D.Tsichritzis, S.Christodoulakis, P.Economopoulos, C.Faloutsos, D.Lee, A.Lee, J.Vandenbroek, C.Woo, A Multimedia Office Filing System , The Tenth International Conference on Very Large Data Bases, 1984.

3. D.Tsichritzis, F.Lochovsky, Data Models, Prentice-Hall, Englewood Cliffs, N.J., 1982.

4. D.Tsichritzis, "OFS: An Integrated Form Management System", Proc. Sixth Int. Conf. on Very Large Data Bases, pp. 161-166, 1980.

ON AN INFERENCE ENGINE FOR EXPFRT SYSTEMS

HO TU BAO

Institute of Computer Science and Cybernetics
Ha noi , Viet nam

ABSTRACT

This paper describes the inference engine COTO , an automated reasoning tool for building expert systems in which the user-friendliness of knowledge engineering is emphasized.

KEYWORDS

Knowledge representation , rule base , fact base, reasoning strategies , inference engine , reasoning explanation .

INTRODUCTION

Expert Systems probably constitute today the "hottest" topic in artificial intelligence and its resultant technology , limited to academic laboratories previously , is now becoming cost-effective and is beginning to enter into industrial applications.

Building an expert system used to be hard and took years from scratch. It required thousands of hours of programming just to put the capability for intelligent behaviour into a computer. Then a long time of developement in which human expertise are added to the underlying program. Finally a period of debugging and fine tuning.

Bringing the scientific results into real-world applications requires the existence of right tools able to structure , to deduce , to explain and to deal with a large amount of knowledge . This ability is exercised with respect to the correctness and the elimination of contradiction.

This paper describes the reasoning mechanisme COTO and deals with the design principles and implementation aspects of an expert system based on this mechanisme.

An Expert System usually consists of two essential parts :

- a Knowledge Base ,
- an Inference Engine .

The Knowledge Base consists of a set of RULES , which present part of the knowledge source of experts in a given domain , and a set of FACTS , which relate to a particular situation to be analyzed . These sets are referred to as the Rule Base and the Fact Base.

To construst a successful knowledge base , the following prerequisites must be met :

- There must be at least one human expert acknowledged to perform the task of defining the set of Rules .
- The primary source for the expert's knowledge is judgement and experience.
- The expert must be able to explain the applications of the special knowledge to particular problems .
- The task must have a well-bounded application domain .

The Inference Engine is a problem-solving program . It has the capacity of learning , structuring and manipulating of knowledge in an intelligent way .It is rather independent with respect to the domain of the knowledge base . With different knowledge bases appropriating with the representation syntax , one can construct many expert systems without modify the inference engine .

The power of an inference engine is characterized primarily by its capacity of manipulating the underlying logical representation of knowledge and also by its flexibility , user-friendliness and speed of reasoning ... Inference engine design may best be considered as an art form in which the chosen design can be implemented from the collection of available artificial intelligence techniques in heuristic search and problem solving .

As an automated reasoning tool for building expert systems , the inference engine COTO has the following features :

- FORTRAN 77 implementation .
- Reasoning with numerical and alphanumerical variables. These variables must be instancied in the moment of deduction .
- Reasoning in forward and backward chaining according to the need of user.
- Reasoning in tri-valued logic : affirmation , negation, ignorance and without limination by Horn clauses .
- Interacting easily with users by quasi-natural language , i.e. readable by somebody not involved in computer science .
- Explaining its reasoning by pointing out various steps in the inference process .

The overall structure of an expert system based on COTO is shown in figure 1. The inference engine is the heart of the system and consists of the following basic components :

- . Rule-compiler : Reads the rule base and builds an internal representation of knowledge.
- . Knowledge acquisition : Adds the knowledge in the fact base , actives and desactives the rules in reasoning.
- . Inference : Reasons in forward and backward chaining.
- . Dialog : Interacts with users in the quasi-naturel language, explains the reasoning process.

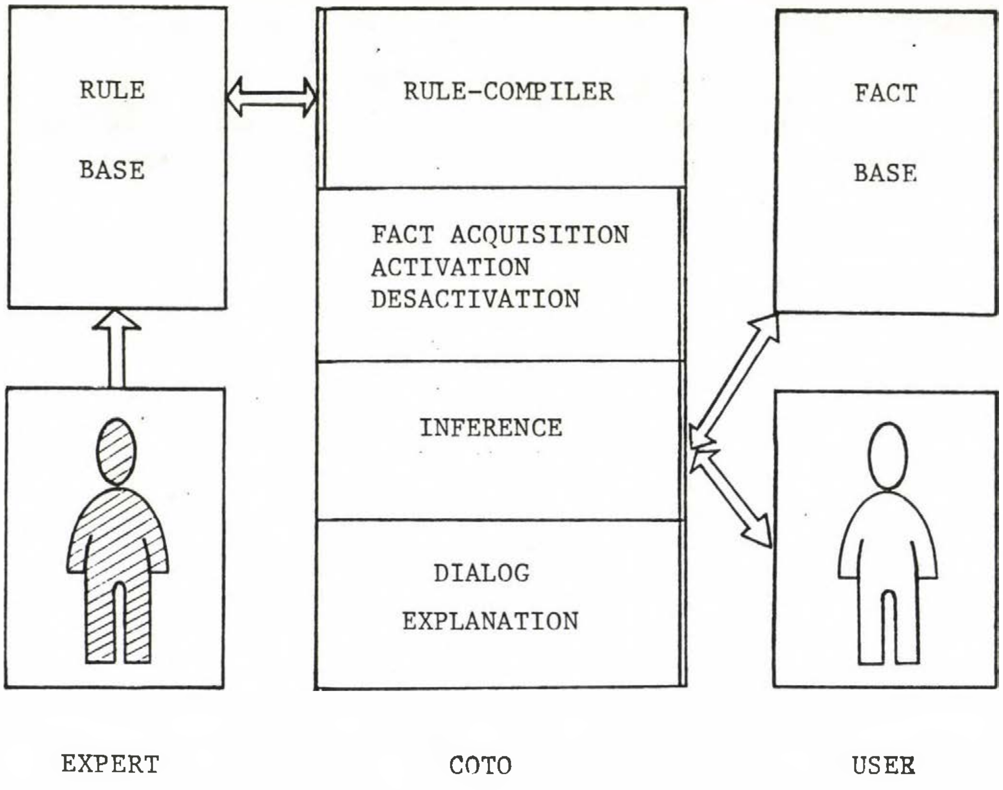


Fig.1. Overall structure of an expert system based on COTO

⟹ Interface

II. REPRESENTATION OF INFERENCE ENGINE COTO

II.1. Knowledge representation

As in almost expert systems , the knowledge is represented in COTO by prodution rules with a simple syntax . To write the rules , one uses only the followings :

"IF" , "THEN" , "AND" , "NOT" , "RULE"   for keywords,

the comparison operators :

">", ">=", "<", "<=", "=", "<>"   for numerical variable,
"==" , "><"   for alphanumerical variable,

and the assigment operators :

"<=="   for numerical constant,
":="   for alphanumerical constant.

The production rules have the form :

```
RULE  k
IF       <premise 1>     AND ... AND  <premise n>
THEN     <conclusion 1>  AND ... AND  <conclusion m>
```

The <premise> and <conclusion> must fit the following syntax

< object 1 > < relation > < object 2 >

where :

< object 1 > may be a proposition , a numerical or an alphanumerical variable or a numerical function given by a name and a list of parameters between two parenthesis .

< object 2 > may be a numerical or an alphanumerical constant, a numerical variable or function .

< relation > may be one of comparison or assigment operators described above.

For instance , the following premises and conclusions are available :

```
   IF  THE MONKEY IS HOLDING THE BANANA

   IF  TEMPERATURE OF PATIENT   >=  40.5

   IF  SUM(LAMBDA1,LAMBDA2,LAMBDA3)  >  0.8

   IF  NOTE(WEIGHT,HIGHT)   <>  PLUS5(AGE)

   IF  THE COLOUR OF FLAG  ==  BLACK

   IF  FORM OF TABLE  ><  CIRCULAR
```

For example , a rule in COTO may be :

```
RULE13

IF      Diagnosis  ==  prime or second breach
  AND   Delta of PGV   <   20
  AND   Plus5(Pprime)  >   PGVmax
  AND   PGVmax         >   40
THEN
        Diagnosis   :=   prime breach
  AND   Procedure   :=   A23
  AND   Tric        <==  Mean(PGVmax,PGVmin)
```

In this rule , "Diagnosis" , "Procedure" are understood by the system as the alphanumerical variables , "Delta of PGV", "PGVmax", "PGVmin", "Tric", "Pprime" as the numerical variables , "Plus(.)", "Mean(.)" as the functions whose values depended upon the values of "Pprime" or "PGVmax", "PGVmin" , and "20" , "40", "prime or second breach" , "A23" as the constants .

One can notice that the negation keyword "NOT" may be anywhere among the words of the proposition . For example , "the monkey is not holding the banana" is equivalent to "not the monkey is holding the banana" , and it will be understood by the system as the negation of "the monkey is holding the banana" .

II.2. Acquisition and structuration of knowledge

The first step in the working process of the system is reading of the rule base . If the production rules are written according to the syntax described above , COTO would have the capacity to learn , to structure the knowledge into its internal form. In reading the rules, a domain specific dictionary is built. It contains all of elementary words expressing the facts by the language of experts. And in the same time, the external rules are restructured internally into their inference network. Each premise or conclusion is considered as an element of state space. The description of each state consists of the name of fact in the dictionary, its role and type (premise or conclusion, operative and connective functions), the sense of fact (negative or positive,thrshold value,adress of associated alphanumerical constant), the pointer points to the next adress, and the situation of premise or conclusion in reasoning.

The semantic of each premise or conclusion is established by the system whenever it is involked during forward or backward reasoning.

The factual facts on the concrete situation are affirmed and added initially in the fact base or in the reasoning process by the interacting with the system . There exists no codification of knowledge and this leading to one difficult problem of knowledge acquisition . The users do not know how enter the facts so that these facts will correspond to the system knowledge . The simple and effective way used in COTO is to make users recognize the system vocabulary by the order of the apparition frequencies of words or the affirmed or deduced facts in the fact base concerning to the situation .

The facts are registered in the fact base in the form of a triplet < object , value , type > . Each time when a fact is affirmed or deduced , an activation and desactivation procedure runs over the rule base for propagating the information and limiting the useless posibilities.

II.3. Strategies of reasoning

Two basic strategies of FORWARD and BACKWARD reasoning are used mutually in COTO according the need of user .

After reading of rules , COTO asks the user for a set of initial facts and for a possible goal to prove.

If there exists a goal to achieve , COTO is in the backward reasoning . It begins by examining a limited set of production rules whose conclusion contain the goal . Then it proceeds to verify the premises of these rules to see which of the goal are satisfied . As the rules are examined in this backward unraveling , some premises are unknown and therefore they become new subgoals. This process is perfomed until the first goal is affirmed or no more rules are activable.

If there is any particular goal in the begin , COTO starts with a set of initial facts and proceeds to invoke the rules in the forward direction . This will continue until no further rules can be invoked .

If no fact or goal is given by the user , COTO tries to ask the "most information" questions determining by the number of apparitions of each premise.

In fact , both MODUS PONENS and MODUS TOLLENS are used to produce new facts or to prevent contradictory reasoning . By accepting the reponse "I don't know" for the questions , COTO functions also in non monotonous logic .

An agenda-driven mechanisme is built which lists the tasks that the system could preform. It provides a good way of choosing the most promising task on each cycle. As the knowledge bases grow, the agenda becomes a particularly significant advantage.

II.4. Dialog and explanation

This plays an important role in the working process of the system. Through dialog , COTO may eventually provide expert advice or a solution to the user's problem , or suplly information that the user is looking for . The user interact with COTO essentially by question answering and vice-versa.

Depending on the variable type in the premise examined and the different situations , the system asks the convenable questions :

- "Do you want to ... ?"
- "Do you think that ... is true ?"
- "Can you give me ... ?"
- "Do you know ..."

The answers are always simple :

- "Y" or "YES" ;
- "N" or "NO" ;
- "!" (I don't know) ;
- a number (for a numerical value) ;
- an order (for an alphanumerical value) .

The user may also request whenever :

- "?" (why) ;
- "h" or "help" (for obtaining the useful information) ;
- "t" or "trace" (for displaying the fact base) ;
- "i" or "insert" (for inserting new facts in fact base) ;
- "s" or "stop" (to stop reasoning) .

The explanation of COTO according to the system status when being asked "?" . It is particularly designed for the reason of making decision.

III. EXAMPLE OF UTILISATION OF COTO

III.1. The rules

We take here a simple knowledge base . This is the tracduction of organigram of diagnosis 5 mm after the injection of security in a PWR 900 MW nuclear core , cf. 1. procedure A0 notes Morori (1982) and Brillon , Janin , Munier (1981) .

```
RULE 1
IF       PPRIME  > 138
  AND    PPRIME  < 160
  AND    DELTA PGV  <  4
  AND    PURGES GV OR CONDENSER ==   NON-ACTIVITY
THEN     DIAGNOSIS IS OVER-ABUNDANT
  AND    PROCEDURE   :=  I3

RULE 2
IF       PPRIME  <=  138
THEN     DIAGNOSIS IS BREACH

RULE 3
IF       PPRIME  >=  160
THEN     DIAGNOSIS IS BREACH
```

```
RULE 4
IF          DELTA PGV  >=  4
THEN        DIAGNOSIS IS BREACH

RULE 5
IF          PURGES GV OR CONDENSER  ==  ACTIVITY
THEN        DIAGNOSIS IS RUPTURE TUBE GV
  AND       PROCEDURE    :=   A3

RULE 6
IF          DIAGNOSIS IS BREACH
  AND       PURGES GV OR CONDENSER  ==   NON-ACTIVITY
THEN        DIAGNOSIS IS PRIME OR SECOND BREACH

RULE 7
IF          DELTA PGV   >=   20
  AND       DIAGNOSIS IS PRIME OR SECOND BREACH
THEN        DIAGNOSIS IS SECOND BREACH

RULE 8
IF          DIAGNOSIS IS PRIME OR SECOND BREACH
  AND       DELTA PGV  <  20
  AND       PLUS10(PPRIME)   <   PGVmax
THEN        DIAGNOSIS IS APRP                        /LARGE BREACH/
  AND       PROCEDURE     :=   A12

RULE 9
IF          DIAGNOSIS IS PRIME OR SECOND BREACH
  AND       PLUS10(PPRIME)    >   PGVmax
  AND       PGVmax  <  40
THEN        DIAGNOSIS IS SECOND BREACH

RULE 10
IF          DIAGNOSIS IS PRIME OR SECOND BREACH
  AND       DELTA PGV  <  20
  AND       PLUS10(PPRIME)  >    PGVmax
  AND       PGVmax      >=  40
THEN        DIAGNOSIS IS PRIME BREACH

RULE 11
IF          DIAGNOSIS IS SECOND BREACH
  AND       TRIC  >=   286
THEN        DIAGNOSIS IS SECOND BREACH IN ENCLOSURE   /OVERHEATING/
  AND       PROCEDURE   :=  A23

RULE 12
IF          DIAGNOSIS IS SECOND BREACH
  AND       TRIC   <  286
  AND       P ENCLOSURE    ==   ELEVATED ANORMALLY
THEN        DIAGNOSIS IS SECOND BREACH IN ENCLOSURE       /COOLING/
  AND       PROCEDURE   :=   A22
```

```
RULE  13
IF         DIAGNOSIS IS SECOND BREACH
  AND      TRIC  <  286
  AND      P ENCLOSURE  ==   NORMAL
THEN       DIAGNOSIS IS VAPOUR TUBE RUPTURE OUT OF ENCLOSURE
  AND      PROCEDURE   :=   A21

RULE 14
IF         DIAGNOSIS IS PRIME BREACH
  AND      DECH VALVE PRESSED OR ASP NORM  ==  LARGE OPEN
THEN       DIAGNOSIS IS DEPRESSED PRIME CIRCUIT
  AND      PROCEDURE   :=   A8

RULE 15
IF         DIAGNOSIS IS PRIME BREACH
  AND      DECH VALVE PRESSED OR ASP NORM  ==  NON-LARGE OPEN
  AND      PPRIME   >   PGVmax
THEN       DIAGNOSIS IS APRP                          /SMALL BREACH/
  AND      PROCEDURE     :=   A11

RULE  16
IF         DIAGNOSIS IS PRIME BREACH
  AND      DECH VALVE PRESSED OR ASP NORM  ==  NON-LARGE OPEN
  AND      PPRIME  <=   PGVmax
THEN       DIAGNOSIS IS APRP                          /LARGE BREACH/
  AND      PROCEDURE   :=  A12
```

III.2. Example of execution

```
COTO

List of available files of rules :

  1. Choice of Miss France
  2. Choice of methods of data analysis software SICLA
  3. Injection of security in the PWR 900 MW nuclear core
  4. Recognition of flags
  5. Recognition of mushrooms

On what file do you want to work ?
3

I read now the rule base ,
Let me a bit of time .
Do you know how interact with COTO ?
If you want to know , please tape "y" ( or "n" )
no


I have read          16   rules in rule base ,
Then I registered    36   words in my dictionary ,
Among them            1   numerical function ,
and                  13   askable goals .


Do you want print the dictionary ( by piece ) ?
no
Please  enter  the  facts  (  $  for END ) ?
$
Do you have a particular goal to achieve ?  (its name or "n")

PROCEDURE == A11

You tell me that : PROCEDURE == A11
OK ! I understand .
For up-dating the working espace , enter the facts , please ?

$
Do you think that  DIAGNOSIS IS PRIME BREACH  is true ?
?

I try to apply all of rules dealing with
DIAGNOSIS IS PRIME BREACH , for example :

RULE 15
IF      DIAGNOSIS IS PRIME BREACH
  AND   DECH VALVE PRESSED OR ASP NORM == LARGE OPEN
  AND   PPRIME   >   PGVmax
THEN    DIAGNOSIS IS APRP                           /SMALL BREACH/
  AND   PROCEDURE  := A11
```

```
Do you think that  DIAGNOSIS IS PRIME BREACH  is true ?
!
Do you think that  DIAGNOSIS IS PRIME OR SECOND BREACH
is true ?
!
Do you think that  DIAGNOSIS IS BREACH  is true ?
!

Can you give me the value of : PPRIME ?
72


You have affirmed me that

Fact 1 :  PPRIME = 72.0   <=  138.0
Then I come to the following conclusion :
After RULE 2 , I can deduce that :

1.  DIAGNOSIS IS BREACH


Can you tell me  PURGES GV OR CONDENSER ,
Is it ?

1.  NON-ACTIVITY
2.  ACTIVITY

1
OK !
You tell me that : PURGES GV OR CONDENSER == NON-ACTIVITY


With the conclusions :

Fact 2 :  DIAGNOSIS IS BREACH
Fact 3 :  PURGES GV OR CONDENSER == NON-ACTIVITY

After RULE 6 , I can deduce that :

1.  DIAGNOSIS IS PRIME OR SECOND BREACH

Can you give me the value of  DELTA PGV  ?
trace

What kind of facts you want see ?

    1. the facts affirmed
    2. the facts deduced
    3. all the facts in the base
3
```

```
BASE OF FACTS

1. PPRIME                                          72 (affirmed)
2. DIAGNOSIS IS BREACH                           true (deduced )
3. PURGES GV OR CONDENSER                NON-ACTIVITY (affirmed)
4. DIAGNOSIS IS PRIME OR SECOND BREACH           true (deduced )


Can you give me the value of  DELTA PGV  ?
1
Can you give me the value of  PGVmax  ?
?

I try apply all of rules dealing with : PGVmax , for example

RULE 10
IF       DIAGNOSIS IS PRIME OR SECOND BREACH   (affirmed)
  AND    DELTA PGV  <   20   (affirmed)
  AND    PLUS10(PPRIME)  >  PGVmax
  AND    PGVmax   >=  40
THEN
         DIAGNOSIS IS PRIME BREACH

Can you give me the value of  PGVmax  ?
71


With the conclusions :

Fact 4  :  DIAGNOSIS IS PRIME OR SECOND BREACH
Fact 5  :  DELTA PGV  =  1    <   20
Fact 6  :  PLUS10(PPRIME)   =    82
Fact 7  :  PGVmax   =  71   >=   40

After RULE 10  I can deduce that :

1.  DIAGNOSIS IS PRIME BREACH


Can you tell me  DECH VALVE PRESSED OR ASP NORM ,
Is it ?

   1.  LARGE OPEN
   2.  NON-LARGE OPEN
2
OK !
You tell me that : DECH VALVE PRESSED OR ASP NORM ==
                                      NON-LARGE OPEN

All right , at last I come to your goal ,
With the conclusions :

Fact 8 : DIAGNOSIS IS PRIME BREACH
Fact 9 : DECH VALVE PRESSED OR ASP NORM == NON-LARGE OPEN
Fact 1 : PPRIME  =  72.0
Fact 7 : PGVmax  =  71.0

After RULE 15 I can deduce that :

1. DIAGNOSIS IS APRP /SMALL BREACH/
2. PROCEDURE  :=  A11

I have responded on your request ,
We try to pass all of possible rules ?
non

Do you want to restart reasoning from one fact affirmed ?
n
Do you want to start another reasoning ?
n


GOOD BYE ! IT'S NICE TO SEE YOU AGAIN !
```

V. CONCLUSION

COTO is a basic tool to build expert systems which incorporates many artificial intelligence techniques . Reasoning automatically in the natural way , it allows the user easily utilise its features without having to be able an expert artificial intelligence programmer.

COTO is now actually applied to various problems :

- Data Analysis : to help the choosing method and interpreting results for the data analysis software SICLA . The interest of the FORTRAN 77 implementation is that COTO can be intergrated to SICLA . Then it is possible to provide new rules after a data analysis , as well as analyzing the the knowledge base itself as data table .

- Pattern Recognition : determining the effective recognition procedure after the data analysis step .

- Biology : Recognizing mushrooms .

REFERENCES

1. Demonchaux E. , Quinqueton J. , " OURCIN 2.1 : Manuel de reference ", Technical Report No 57, Institut National de Recherche en Informatique et en Automatique , Paris , Sep. 1985.

2. Lauriere J.L. , " Representation et utilisation des connaissances ", Technique et Science Informatique, Vol1, N1 and N2, 1982.

3. Rich L. , Artificial Intelligence , Inter.Stud.Ed., 1984.

4. Weiss S.M., Kulikowski C.A., A pratical guide to designing experts systems, Rowman & Allanheld Publishier, 1984.

5. Ho T.B. , Quinqueton J. , " COTO : Moteur d'inference , application en Analyse de Donnees", Research Report , Institut National de Recherche en Informatique et en Automatique, Paris (to appear).

6. Gondran M. Introduction aux systemes experts , Eyrolles, Paris , 1984 .

7. Yasdi R., A conceptual design aid environment for expert-database systems, Data & Knowledge Engeneering 1(1985) .

8. Alty J.L., Coombs M.J., Expert Systems. Concepts and Examples, NCC Publications 1984.

LOW-LEVEL EXTENSIBLE LANGUAGE SYSTEM FOR IMAGE PROCESSING

Andrzej BIELIK, Michał MŁODKOWSKI, Maria PIOTROWICZ

Institute of Biocybernetics and Biomedical Engineering,
Polish Academy of Sciences
KRN 55
00-818 Warsaw, Poland

Abstract

We present in the paper a proposal for a standard set of primitives of picture processing /parallel and sequential/ operations together with means to compose any operation from them. These composition rules take a form of a language. It is built as an extension of the C language. The method of implementation of the system is also essential in our approach. Since we use a special compiler-compiler technique of implementation, we gain extensibility and portability of the system.

I. INTRODUCTION

First of all, we distinguish in a picture processing system three different levels :

1. machine instruction / or operation primitive / level;

2. operation / or composition language / level; and

3. user command / or task / level.

This distinction may look different in different systems depending on what is assumed as primitive in a given system. Access to pixels, some primitive operations on them, and some scanning control primitives are primitives for a sequential processor, whereas for array processor some operations on the whole pictures are primitives. The difference between levels lies in their form rather than contents. While levels 1 and 3 are sets of parameterized commands, level 2 is a set of expressions built according to some composition rules. Decomposition of the system into levels makes it modular and enables its analysis. We deal here only with levels 1 and 2.

The first thing in system analysis is to check whether it contains all three levels. Level 1 must always be present. Thus only the question of level 2 is important for us here. Systems with this level included are versatile in the sense that indefinite number of qualitatively different operations can be expressed in them.

Systems built as libraries of subroutines / e.g. SPIDER / 1 / / in fact lack composition language level and, in consequence, are not versatile. You will always find an operation for which you will have to write a new, special subroutine, even though the library were embedded in a language. The point is that the elements of operations should be embedded in the language rather than operations themselves. Otherwise the language acts as a user command language rather than composition language.

The next important point is the method of implementation of the composition language. Several approaches were taken in published systems.

1. Picture operations primitives embedded in existing general purpose language as procedures /e.g. in C /2/ /;
2. Picture operations primitives and composition language built over existing general purpose language as its extension :
   2.1. by means of preprocessor /e.g. over PASCAL /3/ /;
   2.2. by means of modification of existing compiler /e.g. ALGOL 60 /4/ /.

Although approaches 1 and 2.1 give systems which are easily extensible / new operations can be easily added / and are portable / changed hardware implementation of primitives is met by change of appropriate procedures /, yet they suffer from cumbersome syntax. On the other hand, approach 2.2. ensures excellent syntax, but systems are hard to modify. Therefore we have taken another approach. We use a special compiler-compiler which enables easy modifications and convenient syntax at the same time.

The questions of the set of primitives and the composition language themselves are, of course, most important design problems. We have chosen the usual way of extending general purpose language with pictorial primitives and mechanisms. We have taken the C language for that purpose.

Next sections will be devoted to pictorial extensions as well as compiler-compiler description.

II. PICTURES

Pictures are represented as 2D arrays of numbers. They can be of any size and any gray value range / up to 2 bytes / including binary pictures. They can be stored in main memory, display memory, or mass storage.

Some picture analysis tasks / e.g. the intelligent recognition / require processing of parts of pictures only. Typically, they are windows and grids. The grid is a set of pixels evenly spaced over picture. The grids are usefull for rough picture processing.

III. PICTURE SCANNING OPERATIONS

The picture operation is an arbitrary function with pictorial arguments and/or results. We distinguish a broad class of most often used operations and call them scanning operations. They are those which can be written in the form of the following function :

$$f\ /x,\ S_x,\ f\ /p/x/,\ \dots\ /\ /,$$

where:

x is the pixel from the whole picture or its part,

S_x is the neighbourhood of pixel x,

p /x/ is a pixel preceding x / relatively to some order /. If f depends on x only, then it is called point-wise operation; If it depends on S_x then it is called local operation; If it depends on f /p/x/, ... / then it is called sequential operation / because to compute its value in a given pixel, its value in the preceding pixel is necessary/. If f does not depend on f /p/x/, ... / then it is called parallel operation / because its value in any pixel does not depend on the computation order /. The parallel as well as sequential operations could be point-wise or local.

IV. PRIMITIVES OF PICTURE SCANNING OPERATIONS

Picture operation of the form described in previous section can be divided into 3 independent processes : control of picture scanning, access to pixel / or to its neighbourhood / and the proper computation of pixel value / or its neighbourhood /.

First and second processes take most of the execution time of the whole operation / when they are implemented on universal computer /. The idea is to supply the user with the above processes as primitives by means of which the arbitrary picture operation can be composed. Processes of scanning control and access to pixel can be divided into 2 groups : concerning whole picture / executed only once for whole picture / and concerning pixel / executed for every pixel /. In consequence, a user obtains the following procedures :

- initiation of scanning / control and access to pixel / for whole picture;
- transition to successive image pixel; and
- access to current pixel or to its neighbourhood.

The following standard kinds of scanning are realized :

- sequential
- lexicographical / row-wise from left to right and from top to bottom /
- reversed lexicographical / row-wise from right to left

and from bottom to top /.
The lexicographical scanning is most effective because it coincides with image alocation in computer memory. Therefore it can be used to simulate parallel scanning / provided that the input and output images are distinct /.

All of the above primitives are realized in following versions : for the whole image, window and grid; for different types of images; and for different types of picture store.

The proper pixel processing is realized through direct use of processor instruction or language in which the procedures are embedded.

The above approach has made it possible to attain modular extensibility as well as time-effectiveness of our system. The time-effectiveness is achieved owing to simplification of control and access to pixel by means of extracting the part which concerns a whole picture, and due to possibility of execution of control processes and access to pixel only once for many operations on the picture.

V. ELEMENTARY PICTURE OPERATIONS

Picture operations can be functionally divided into the following groups :

1. Access and change of value

1.1. Reading, writing and copying of : pixel, window and whole image.

1.2. Input/output.

2. Picture processing / picture onto picture /

2.1. Arithmetical-logical pixel-wise

2.2. Arithmetical-logical local / e.g. convolutions /

3. Change of picture form

3.1. Onto the picture with other characteristics / spatial resolution, dimensions, gray values range, position / e.g. rotations, shifts / /

3.2. To the non-pictorial form / e.g. lists /

3.2.1. Feature calculation

3.2.1.1. Histograms analysis

3.2.1.2. Regions localization

3.2.1.3. Other

3.2.2. Other

3.3. From the non-pictorial to pictorial form / objects generation /.

Besides primitives of image scanning we include to the system some operations from above groups as elementary ones. Namely those which are not picture scanning operations, some of scanning operations which are often used, and some domain specific operations. Among them there are edge and region extraction operators.

VI. SYNTAX OF COMPOSITION LANGUAGE

We give brief review of picture processing extensions to C language.

1. Declarations

One can declare : pictures /PIC/, windows /WINDOW/, grids /GRID/, neighbourhoods /LOC/, and pixels /PIX/. Pixels are structures with fields : X, Y, VAL. All subsequent occurences of identifiers with suffix "_id" stand for names of appropriate objects.

2. Data access

One can access :

pixel in picture :	pic_id/x,y/
window in picture :	pic_id W/m,n/ AT/x,y/
grid in picture :	pic_id G/m,n/ AT/x,y/
element of pixel :	pix_id.x, pix_id.y,pix_id
element of neighbourhood :	ne_id/x,y/, ne_id.pix_id

3. Scanning statements

Parallel scanning : FOR / par_list/ statement

Reverse "parallel" scanning : REVERSE FOR / par_list/ statement

Sequential scanning : FOR / seg_list/ /init; test; iner/
statement

where

"par_list" is a list of the following expressions :

```
 ALLloc_id    IN domain,    / for local/
 ALLpix_id    IN domain,    / for point-wise /
```

"seg_list" is a list of the following expressions :

```
 loc_id       IN domain,    / for local /
 pix_id       IN domain,     / for point-wise /
```

"domain" is one of the following expressions :

```
  picture
  window_id    OF   picture
  grid_id      OF   picture
  mask_id      OF   picture
```

All pixels and neighbourhoods contained in the list move together during picture scanning.

4. Expressions

Expression may have one of the following forms :

```
    unop   picture,
    pic 1  binop   pic 2 ;
```

where : "unop" and "binop" are standard C operators ;

"binop" may also be "**", which means convolution.

VII. IMPLEMENTATION

Language is being implemented by means of special compiler-compiler. It accepts as its input language description which consists of BNF syntax rules with attributes and translations appended to them. On output one gets complete compiler with LALR/1/ parser and code generator consistent with defined translations. One can easily modify language description and obtain new compiler.

Implementation is carried out on an experimental image processor DIPP which is under development in our laboratory.

References

1. H.Tamura, S.Sakane, F.Tomita, N.Yokoya, M.Kaneko, K.Sakaue, Design and Implementation of SPIDER Transportable Image Processing Software Package, in: Computer Vision, Graphics, and Image Processing 23,1983, pp.273-294.

2. J.Piper, D.Rutovitz, Data Structures for Image Processing in C language and Unix Environment, in: Pattern Recognition Letters, vol.3, N^{o}2, March 1985, pp.119-129.

3. L.Uhr, A Language for Parallel Processing of Arrays, Embedded in PASCAL, in: M.J.B.Duff, S.Levialdi, eds, Languages and Architectures for Image Processing, Academic Press 1981, pp.53-87.

4. S.Levialdi, et al, On the Design and Implementation of PIXAL, a Language for Image Processing, ibidem, pp.89-98.

Microcomputer monitoring of the side-effects in Hungarian pharmacological study

M. Csukás[x], E. Farkas[xx], A. Krámli[xxx], G. Maróti[xx]
and J. Soltész[xxx]

[x] National Institute of Cardiology

[xx] Richter Gedeon Pharmaceutical Company

[xxx] Computer and Automation Institute, Hungarian Academy of Sciences

A pharmacological study - comparing two preparates and a placebo - is being carried out for the Richter Gedeon Pharmaceutical Company. The study - according to the international standards - is based on fixed sample statistical methods. However the side effects are monitored by using sequential procedures (for mathematical backgrounds cf. e.g. "Restricted Sequential Procedures", Armitage, P.,Biometrics, 16, 9-26). The sample size is 2500 patients, and about 50 side effects are continuously being monitored. The structure of data is described in the talk "Mikrocomputer-based special medical information system" (Kerékfy, P., Kiss, A., Ratkó, I., Ruda, M.). Here we shall point out only one peculiarity of the monitoring problem.

For each side effect two files were constructed: the first one contains records on the patients suffering from the given side effect and taking the first preparate or the placebo while the second file contains records on the patients suffering from the given side effect and taking the second preparate or the placebo. The two types of files were processed separately. Further on - for the sake of simplicity - our considerations refer to one of the above constructed files.

The records of the files are sorted on ascending key, where key is the time interval from the beginning of the treatment until the first occurrence of the given side effect. This arrangement is the appropriate one for the sequential procedure.

The goal of the sequential procedure is to determine whether one of the two preparates or the placebo causes a given side effect with greater probability. The probability in question is unknown, and its value is irrevelant, because the sequential procedure omits the "indifferent cases" i.e. the patients who do not suffer from the given side effect.

We assign to the ith record (on the sorted file) the value r(i) where

$$r(i) = \begin{cases} +1 \text{ if the ith patient takes the preparate} \\ -1 \text{ if the ith patient takes the placebo} \end{cases}$$

According to Wald's method, if the values r(i) form a random sequence and the probabilities of the occurence of the given side effect are π_1 and π_2 when the patient takes preparate and placebo, respectively,

then $w(n) = \sum_{i=1}^{n} r(i)$ is a random walk which steps +1 with probability

$$\vartheta = \frac{(1-\pi_1)\pi_2}{(1-\pi_1)\pi_2 + \pi_1(1-\pi_2)}$$

and -1 with probability $1-\vartheta$.

Testing the O-hypothesis $\pi_1 = \pi_2$ is equivalent to testing the O-hypothesis $\vartheta = 1/2$. In the practice instead of testing $\vartheta = 1/2$ we always compare two simple hypotheses $\vartheta = \vartheta_1 > 1/2$ and $\vartheta = \vartheta_2 = 1 - \vartheta_1 < 1/2$. The ratio $\vartheta_1 / \vartheta_2$ can be given on the basis of medical consideration.

For given ϑ_1 and ϑ_2 the general scheme of the sequential procedure looks like as follows: for given probabilities of the errors of first and second kind (α and β) the coefficients a and b can be computed:

$$a = \frac{2 \log \{(1-\beta) / \alpha\}}{\log \{\vartheta_1 / (1-\vartheta_1)\}}$$

$$b = \frac{2 \log \left\{ \frac{1}{2} \vartheta_1^{-\frac{1}{2}} (1-\vartheta_1)^{-\frac{1}{2}} \right\}}{\log \{\vartheta_1 / (1-\vartheta_1)\}}$$

We accept the hypothesis ϑ_1 if

$$\min_{w(n) > a+bn} n < \min_{w(n) < -a-bn} n$$

i.e. the random walk w(n) hits the upper boundary a+bn earlier than the lower boundary -a-bn. Figure 1 illustrates the behaviour of w(n) for one of the side effects and for the first preparate

For a reasonable choice of the parameters e.g. ϑ_1=0.7, α=0.025 and β=0.05 the coefficients of the boundary lines are a=8.59 and b=0.2058. So in our example the sample size is not large enough to accept any of the two hypotheses.

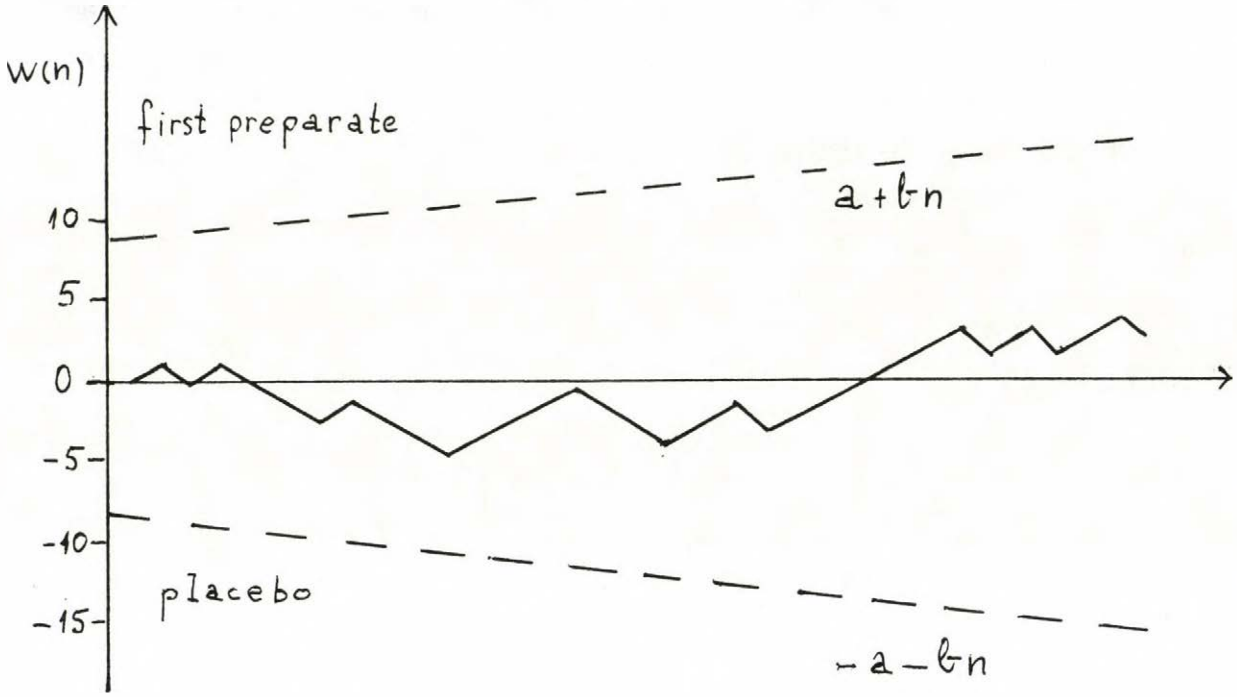


Figure 1.

A technical remark: the treatment begins for different patients at different times, therefore all "side effects files" are to be rearranged when a next step in the sequential analysis is needed.

INTERACTIVE SYSTEM FOR CREATING MAPS IN ARCHAEOLOGY

Dimcho Ivanov Dimov

Institute of Mathematics
Sofia, Bulgaria

In their activities the archaeologists used maps for different purposes - for illustration and for exact registration of archaeological finds. These maps are used for making inferences for the progress of mankind during the different stages of its existence.

This paper describes a cartographic system intended for assisting the archaeologists in creation of maps and plans of the terrain of archaeological investigations.

Computers have come in all spheres of human activities and changed many of the traditional methods of work in different fields of science and practice. One of them in which computers have big application is cartography. The using of computers in cartography changed fundamentally its production. Computers and existing graphical devices enable many of the processes in the production of maps to be automated - from the information insurance till the automatic drawing of the maps.

One of the branches of science, using maps is archaeology. The archaeologists used maps for different purposes. Some of the maps are designed for illustration of definite archaeological material, and the others - for exact registration of archaeological finds. The maps used for these purposes are in different scales and sizes. For instance -

a map of the earth with places of the ancient archaeological finds related to the primitive society, or a plan of an ancient village with noted ruins of buildings, single objects and other elements.

DESCRIPTION OF THE SYSTEM

The described cartographic system is designed for creating maps and plans of regions of archaeological investigations and is working on a personal computer IBM PC. The necessary graphical devices are - plotter and digitizer.

1. Sources of information

There are several sources of information for a graphic data base - cartographic maps, aircraft photographs, landsat images, statistical information, geodetic surveyings, thematic data. Because of the nature and designation of the system we used only some of the above mentioned sources:

- existing cartographic maps.

For creating illustrative maps as base maps are used existing ones on which the desired archaeological objects are drawn. Another way of using existing maps is to obtain from these maps positional information for desired objects with the help of digitizer and to use this information for creating new ones. This method is very useful when the number of desired copies of these new maps is large.

- cartographic data received from measurements made on the terrain of archaeological investigations.

These are the coordinates of geographical objects of the area, received by geodetic surveyings. This information is used for making plans of the investigated region.

- archaeological data.

It is information for the geographical location and archaeological interpretation of the examined objects. This information is used for showing on maps desired archaeological objects located on the territory of a definite district, country and etc. For plans the desired archaeological information is for the exact location and meaning of every single object found on the terrain of the investigations.

2. Data base

The data base used by the system is different from the similar ones used in many other cartographic systems because of its archaeological destination. In the data base there is information for two different types of objects - geographical and archaeological. Everyone of them has its features and is represented with both spatial and aspatial information. The spatial information is a set of coordinates of points in Cartesian system. Aspatial is all other information, which is in relation to defined spatial data, but itself it has no positional meaning. The aspatial information is stored in the data base as relations, the spatial - with the coordinates of representing points.

In the data base there are entities for geometric and cartographic elements. The way of representation of these elements is described below.

Geometric elements:

- point - with x,y,z coordinates.

- straight line - with coordinates of two different points x,y,z and a,b,c.
- curve - with coordinates of a set of points put in order.
- circle - with radius
  x,y,z - centre coordinates
  a,b,c - normal vėctor.
- plane - with x,y,z - coordinates of a pass point, and a,b,c - normal vector.
- polygon in a plane - with one or a few connected curved lines, representing its boundary.

Cartographic elements, representing geographical and archaeological objects:

- interpreted as points.

They are depicted with figures centred with the corresponding points. The sizes and forms of these figures presented some of the aspatial information associated with the treated cartographic or archaeological objects. Such objects are summits, towns and villages, archaeological centres and so on. In the data base the information kept for such objects consists of the type, specifying the real object, its name, and a pointer to the coordinates of the representing point.

- depicted with lines.

These are rivers, roads, railroads, canals, transmission lines and etc. The type of line with which such cartographic element is depicted shows the essential information for the respective real geographical or archaeological object. In the data base for such element there is information for the substance of the real object, its name, and a pointer to the set of points representing it.

- regions.

These are districts, lakes, orchards, vineyards,woodland and etc. They are depicted with their boundaries and the way of shading, if any, shows the type of the represented real object and some of the aspatial information associated with it. The necessary information for such object is the type and name of the real object, and one or more pointers to the set of points representing it.

3. Data input

We used a subsystem, which makes an incoming control and initial processing of the input data, after which the information is stored in the data base.

For input of the non-positional information we used the keyboard of the computer, and for the positional we have two ways - the keyboard and the digitizer. The x,y coordinates of spatial data from the existing maps are received with the digitizer, while the third coordinate z - altitude above sea level - is typed on the keyboard. Its value is the value of horizontal line passing through the digitizing point. The information for the point elements is stored in the data base without further processing. For the linear and polygon elements we used a subroutine for reducing the number of representing points in XY plane, as the allowable tolerance, which specifies how much the precision of the representation should be relaxed, is given by the user. After this the data is stored in the data base.

For the cases of making plans of small areas the positional information is entered from the keyboard of the computer. It is received from geodetic surveyings made on the terrain of the archaeological object. Because of the small sizes of the investigated area it is not necessary to use some special cartographic projection. All the coordinates are in Cartesian system. For centre of the system is used an arbitrary, non-changeable object on the terrain.

The sequence of defining the elements is interactive and the user can change the representation of some parts of the visualized element. For representation of linear and polygon elements are used sets with different number of points, as the connection of points is with segments or with smooth curved lines passing through the points. The way of connection depends on the nature of the objects. If such received picture of the linear or polygon element differs from the form of the real object, new points are taken and the interpolation is done again. The procedure is repeated until the drawn image is satisfactory. After that the information is stored in the data base.

For indicating the archaeological finds is used a set of beforehand defined figures, as the set can be changed. Any of the figures in the set can be changed or deleted, and new figures can be added. The representation of the new figures is given by the user with the help of the graphic cursor or digitizer. The data for every graphic figure is stored in the data base as a structure and when this figure is depicted the information from the structure is modified according to the desired scale.

4. Data processing

Every system must help the user when working with it.

He must have an elementary access to the information in the data base and appropriate methods for its processing, which includes an elementary transformations of data and some more complex processings. The described system can easily extract information from the data base according to specified conditions, which can be used for easy creating the thematic maps. The elementary transformations are needed in most of the cases and include the changing of the scale, coordinate system and cartographic projection.

The more complex processings include possibilities for overlay or logical intersect of spatial data sets having the same geographic coverage, creating one data set with complex attributes, also calculating various spatial relationships / adjacency, linkages/ not stored evidently in the data base. Using the Z - coordinates / altitude above or below sea level/ the isolines can be drawn, and for small areas to draw the surface of the investigated terrain in 3-D coordinate system. There are also possibilities for measuring areas, lengths and distances.

5. Application

The described cartographic system is intended for assisting the archaeologists both in propagating the archaeological information by means of illustrative maps, and in facilitating their own work during the investigations. The plans with the exact registration of the archaeological finds are very useful for further investigations on the same places.

References

1. Nagy, G. and S. Wagle, "Geographic Data Processing", ACM Computing Surveys, Vol. 11, No. 2, June, 1979.

2. Newman, W.M. and R.F. Sproull, Principles of Interactive Graphics, 1979.

3. Uno, S., and Matsuka, H. A General Purpose Graphic System for Computer Aided Design, 6th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH, 1979.

STAR
A QBE-oriented Database Management System

Margareta Draghici
Radu Bercaru, Elena Saftoiu,
Vlad Mihail, Tatiana Dabija

Research Institute for
Computers Technique and
Informatics

Bucharest, Romania

Tiberiu Galos

Computing Centre

Oradea, Romania

Abstract

The paper contains an overview of a relational database management system called STAR, which is being implemented on IBM PC compatible Romanian microcomputers, with a special accent on the solutions for query description and evaluation.

1. STAR - Objectives and general architecture

The large scale production and use of micro and professional personal computers have determined the necessity for versatile software to cope with the large diversity of user requirements.

In the field of database management, progress has been very fast and systems like dBASE II, Rbase, Knowledgeman, Data Base Manager II and many others have quickly received wide acceptance.

A distinct class of users, especially oriented towards personal computers are the so-called "non-professional" users whose main concern besides good performance, is with flexible operation and ease of use.

The relational approach has been adopted, to some extent,for most of the DBMS implementations on microcomputers.STAR is no exception to this trend and we think that building it around a Query-by-Example-like language is an important step towards the above mentioned goals.

So, STAR is a relational database management system designed for the FELIX PC (IBM PC compatible) 16 bit Romanian microcomputers. Nevertheless, the implementation is intended to have a high degree of portability, the system, written mostly in "C", being meant to work under MS-DOS as well as UNIX operating systems.

The STAR language follows the Query-by-Example specifications [1,3]; it is relationally complete and combines,under uniform syntactic patterns, query, data definition and manipulation facilities, with security and integrity constraints specification.

The language has high expresiveness and provides a simple and flexible interface, based on full screen operations, easy to learn and use by non-professionals.

Getting familiar with the system takes no more than 2 or 3 hours:

- a database is constituted as a set of relations

- at interface level, relations are represented as tables

- table headers of the relations involved in a database operation are displayed on user request

- queries are formulated by filling in the tables on the screen; the syntax is very simple and error risk is minimized

- the answer to a query is also a relation, presented in tabular form

STAR provides great flexibility in screen management; relations can be displayed on, deleted from, scanned and moved on the screen by means of functional keys or special commands.

While database definition and loading is performed interactively, STAR also allows for batch-type operations to quickly load or recover large amounts of data stored as standard files.

Users may access information concerning the database structure and are provided with facilities to specify data reorganization, to create snapshots and views.

Database integrity is enforced by a general password mechanism, as well as local constraints, on relation or record level, specified in the STAR language.

Integrating the above mentioned characteristics, STAR could become the ideal tool for non-professionals working with medium sized database applications on microcomputers, the kernel of a complex system for office automation.

The overall STAR architecture is illustrated in figure 1.The organization of the system, in its main components, follows the general principles and criteria used in the design of relational database management systems [4,5]. The peculiarities of the adopted solutions are determined by the computing system on which STAR is being implemented (16 bit microcomputers) and the chosen implementation language ("C") which, according to its specific features, is a bias for some design decisions [7].

The graphical representation in fig. 1 highlights the data flow (marked by arrows) through the system, the control flow being inherent in the structure of concentric rings. Each ring designates a depth level of the structure; the components (modules) forming one ring can "see" i.e. access only modules situated on inner rings (levels).

In the following sections, a model for the formal description of STAR (QBE) queries, as well as an algorithm for query evaluation are presented. The model naturally leads to the generation of the algorithm.

2. The Model

Translation of a STAR query into a tuple calculus expression [4] is staightforward. This advantage is taken into account in the formal description of the query.

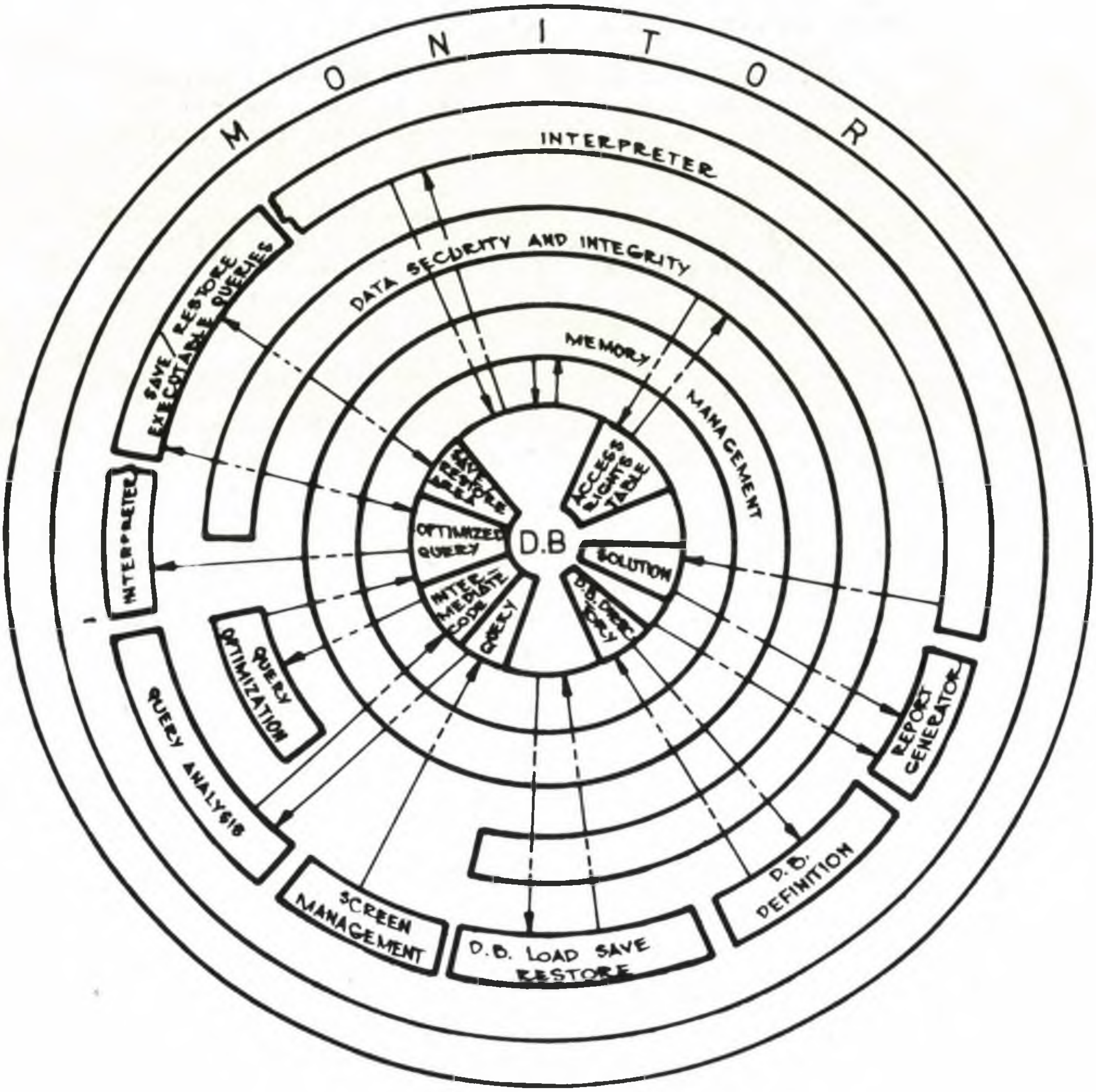


FIG.1. STAR ARHITECTURE

In order to concentrate on the essence of the proposed model, some simpilifying assumptions are made, which do not hamper natural generalization:

(i1) the query includes no condition boxes

(i2) expressions are avoided

(i3) as for the output operators, not their type, but only the output relations they generate are of interest.

For the same reason we use examples to illustrate theoretical statements.

A relation is designated by the letter R followed by an index. Relation attributes are reffered by A,B, ... a.s.o.

Example 1

R1	A	B
s	P.x	<y

R2	A	B
t	<x	P.y

During query analysis, according to [4], one tuple variable, s and t respectively, is associated to each of the two lines. The conditions are isolated:

```
   |
   | c1:  s.B < t.B     and
(1)|
   | c2:  t.A < s.A
   |
```

as well as the output operators:

```
   |
   | o1:  print s.A
(2)|
   | o2:  print t.B
   |
```

Also according to [4], the query evaluation algorithm is the following:

```
     Range R1:s
         Range R2:t
             if c1 and c2 then
                          do:
(3)                          exec o1;
                             exec o2;
                          exit;
         endrange t;
     endrange s;
```

The relations generated by the two output operators can be defined as:

$$(4)\left|\begin{array}{l} o1:\ \{a: \exists\ (s,t) \in R1 \times R2,\ (a=s.A) \wedge c1 \wedge c2\ \}, \\ \\ o2:\ \{a: \exists\ (s,t) \in R1 \times R2,\ (a=t.B) \wedge c1 \wedge c2\ \}. \end{array}\right.$$

We call "query context", the set {s,t} of tuple variables, and refer to it by X.

Assigning values to each tuple variable in X, we get a "context value" denoted by (X).

As results of query analysis, we get:

(a1) The set of tuple variables:

s: Range R1,
t: Range R2 (in ex. 1)

(a2) The set of conditions specified in the query lines, see (1); the set of tuple variables occuring in the expression of a condition is called condition context. We have, for example 1 :

$$(1')\left|\begin{array}{l} c1:\ s.B < t.B\ ,\ \text{context}\ c1:\{s,t\}, \\ \\ c2:\ t.A < s.A\ ,\ \text{context}\ c2:\{s,t\}. \end{array}\right.$$

(a3) The set of output operators applied to fields in the lines, see (2); a context is defined for output operators as well

$$(2')\left|\begin{array}{l} o1:\ \text{print}\ s.A\ ,\ \text{context}\ o1:\{s\}, \\ \\ o2:\ \text{print}\ t.B\ ,\ \text{context}\ o2:\{t\}. \end{array}\right.$$

Scanning all values of a query context X is done by nesting the loops corresponding to the tuple variables in X. The order of loop nesting defines an ordering relation ">" on the set X. In (3) we have s > t, but we should notice that (4) is invariant no matter what order (s > t or t > s) is chosen.

It is necessary that:

(c1) for each condition c in the query, the restriction of the ordering relation > to the condition context (context c) is a total ordering relation and in the evaluation algorithm, c is placed in the loop corresponding to the tuple variable $\min_{>}$ context c or in a loop included in $\min_{>}$ context c.

(c2) for each output operator o, the restriction of the ordering relation > to the operator context (context o) is total and "the highest" location of o is in the loop $\min_{>}$ context o.

For the evaluation of conditions and output operators in a query, (c1) and (c2) are necessary conditions.

Furthermore, the loop nesting ordering relation > should obey the following restrictions:

(c3) the graph of the relation must be a tree, i.e. one single variable r must exist, so that $\exists s \in X \quad s \not> r$, the tree root and for any $s,t \in X \setminus \{r\}$ there exists $\min_{>} \{s,t\}$.

Condition (c3) is of constructive nature and its meaning is to be revealed later on.

(c4) Relation > on X must be "minimal" with respect to conditions (c1), (c2) and (c3), i.e. one of these conditions should be violated whenever any pair (s,t) is removed from the relation graph.

An ordering relation on X satisfying conditions (c1), (c2) and (c3) is called "tree coverage". Starting from a tree coverage we define the "execution tree" by the following:

(d1) the set of nonterminal nodes is the set X of tuple variables

(d2) the set of terminal nodes is the set of conditions and output operators

(d3) the dependencies between nonterminal nodes are determined by the tree coverage (d4) a terminal node c/o is subordinated to the tuple variable min context c/o

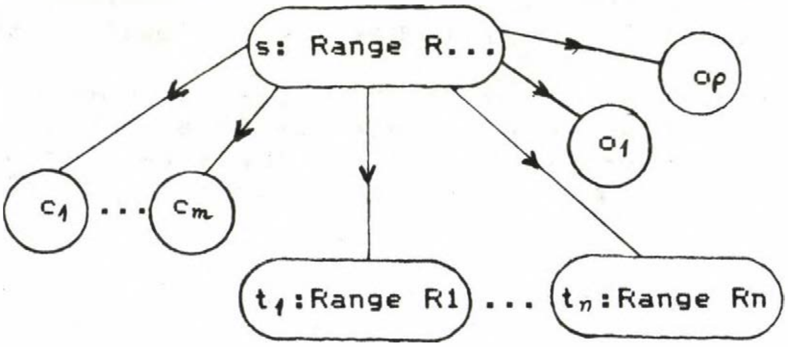
(d5) an ordering relation is enforced between the sons of a nonterminal node

(d5.1) the conditions c come before the first nonterminal node

(d5.2) the output operators o come after the last nonterminal node

3. The Evaluation Algorithm

If, when scanning the execution tree in root-left-right order, for every nonterminal node s



we generate the sequence

```
    Range R:s
        if not $c_1$ then next s;
        ...
        if not $c_m$ then next s;
        Range R1:$t_1$
        ...
        endrange $t_1$;
(5)     ...
        Range Rn:$t_n$;
        ...
        endrange $t_n$;
        exec $o_1$;
        ...
        exec $o_p$;
    endrange s;
```

the resulting algorithm ensures the evaluation of each condition/output operator for all possible values of its context. Hence the meaning of condition (c3) which quarantees the assignment of some value to a tuple variable in all the loops included in the loop associated with the variable.

Yet, algorithm (5) does not provide for the generation of output relations according to the requirements in (4). Some extra conditions are necessary.

Before giving the formal expression of the predicates tested by these conditions, let's see algorithm (5) plus above mentioned conditions applied to example 1.

```
          Pred1(c1):=false;
          Range R1:s
               Pred1(c2;s):=false;
               Range R2:t
                    if not c1 then next t;
                    if not c2 then next t;
(*)                 Pred1(c2;s):=true;
                    exec o2;
               endrange t;
               if not Pred1(c2;s) then next s;
(**)           Pred1(c1):=true;
               exec o1;
          endrange s;
```

where c1 and c2 are notations for nonterminal nodes s and t respectively, and Pred1 is a predicate associated to a nonterminal node c_K .

In algorithm (5) for each nonterminal node c_K we can define its context composed of tuple variables corresponding to nodes c_i,c_i $>c_K$, and denoted by X_K . The current values of tuple variables in X_K constitute (X_K), the current context.

Then Pred1(c_K,(X_K)):X_K --> {true,false}

If **> is an ordering relation determined by the root-left-right linearization of the execution tree, then the meaning of predicate Pred1(c ,(X_K)) should be the following: "(X_K) can be extended to a value (X) in which all the query conditions, c, are satisfied".

The expression of Pred1 changes according to the characteristics of the execution tree.

For example 1 we have:

$\text{Pred1}(c2;s): \exists t \in R2, c1 \wedge c2$
$\text{Pred1}(c1): \exists s \in R1, \text{Pred1}(c2;s) = \exists (s,t) \in R1xR2, c1 \wedge c2.$

If, in example 1, we define

$\text{Pred2}(t;s): c1 \wedge c2$
$\text{Pred2}(s): \text{Pred1}(c1;s),$

we get

$\text{Pred1}(c1): \bigvee_{s \in R1} \text{Pred2}(s) = \exists s \in R1, \text{Pred2}(s)$ and

$\text{Pred1}(c2;s): \bigvee_{t \in R2} \text{Pred2}(t;s) = \exists t \in R1, \text{Pred2}(t;s).$

It can be noticed that Pred2 is obtained from Pred1 by "setting free" the corresponding tuple variable. Pred2 is the condition satisfied by the algorithm at the time the output operator of the respective loop can be executed. Pred2(t;s) corresponds to step (*) in (5) and Pred2(s) corresponds to (**).

By means of Pred2 we have:

$$(4') \left| \begin{array}{l} o1: \{a: \exists s \in R1, (a=s.A) \wedge \text{Pred2}(s)\} \\ o2: \{a: \exists (s,t) \in R1xR2, (a=t.B) \wedge \text{Pred2}(t;s)\} \end{array} \right.$$

In example 1 the expressions of Pred1 and Pred2 are simple since the relation > is total on X. In other cases the expressions get more complicated. Instead of a formal analysis two typical examples are given.

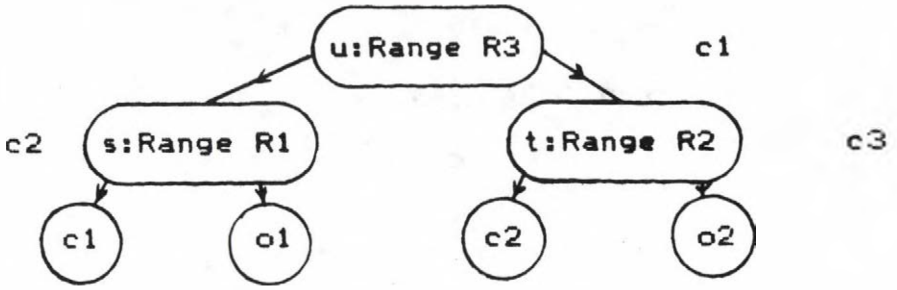
Example 2

R1	A	B
s	P.	<$\underline{x}$

R2	A	B
t	<$\underline{y}$	P.

R3	A	B
u	$\underline{x}$	$\underline{y}$

The execution tree is



where c1, c2 and c3 are nonterminal nodes and > is no longer a total ordering relation on X.

```
c1: s.B < u.A , context c1:{s,u}
c2: t.A < u.B , context c2:{t,u}
o1: print s.A , context o1:{s}
o2: print t.B , context o2:{t}
```

The following algorithm is generated:

```
Pred1(c1):=false;
Range R3:u
    Pred1(c2;u):=Pred1(c3;u):=false;
     Range R1:s
          if not c1 then next s;
          if not Pred1(c3;u) then
               Range R2:t
                   if not c2 then next t;
                    Pred1(c3;u):=true;
                    exec o2;
               endrange t;
          if not Pred1(c3;u) then next u;
           Pred1(c2;u):=true;
           exec o1
       endrange s;
       if not Pred1(c2;u) then next u;
 endrange u;
```

We have:

Pred1(c3;u): $\exists$(s,t) $\in$ R1xR2, c1 $\wedge$ c2
Pred1(c2;u): $\exists$(s,t) $\in$ R1xR2, c1 $\wedge$ Pred1(c3,u)
Pred1(c1): $\exists$u $\in$R3, Pred1(c2;u).

Loop t is included within loop s, but its execution is conditioned by Pred1(c3;u) (loop t is not executed for all the values of tuple variable s).

Example 2´ (example 2 without output operator o1)

The generated algorithm is

```
Pred1(c1):=false;
Range R3:u
     Pred1(c2;u):=Pred1(c3;u):=false;
     Range R1:s
          if not c1 then next s;
          Pred1(c2;u):=true;
          exit;
     endrange s;
     if not Pred2(c2;u) then next u;
     Range R2:t
          if not c2 then next t;
          Pred1(c3;u):=true;
          exec o2;
     endrange t;
     if not Pred1(c3;u) then next t;
endrange u;
```

We have:

$\text{Pred1}(c3;u){:}\ \exists (s,t) \in R1 \times R2\ ,\ c1 \wedge c2$
$\text{Pred1}(c2;u){:}\ \exists s\ ,\ c1$
$\text{Pred1}(c1){:}\ \exists u\ ,\ \text{Pred1}(c3;u) \wedge \text{Pred1}(c2;u)$

Since now, node c2 has no output operator, we call it "unproductive" and handle it as a condition subordinated to node c1.

Without going into further details, we state the very important fact that the expressions of Pred1 and Pred2 for examples 2 and 2´ enable formal correctness proofs for the generated algorithms, i.e. it can be demonstrated that relations generated by the output operators are of type (4) according to the requirements of the algorithm presented in [4].

Final Remarks

The model and the evaluation algorithm described in the paper represent a theoretical background for the query decomposition procedures implemented in STAR.

Further work is dedicated to the generalization of solutions for all the STAR language features and to the design and implementation of an optimization component [6] based on this strategy. A first version of the STAR system is due by the end of 1986.

References

1. M.M.Zloof - Query-by-Example: A Database Language. IBM Systems Journal, vol.16, no.4, 1977

2. M.W.Blasgen, K.P.Eswaran - Storage and Access in Relational Databases. IBM Systems Journal, vol.16, no.4, 1977

3. M.M.Zloof - QBE/OBE: A Language for Office and Business Automation. Computer, no.5, 1981

4. J.D.Ullman - Principles of Database Systems. Computer Science Press, 1980

5. C.Delobel, M.Adiba - Bases de donnees et systemes relationnels. BORDAS, 1982

6. M.Draghici, R.Bercaru, E.Saftoiu, V.Mihail, T.Dabija - Criteria and Solutions for Query Optimization in STAR (in Romanian). Research Report, 1985

7. R.Bercaru, E.Saftoiu, V.Mihail, T.Dabija - STAR: System Architecture (in Romanian). Research Report, 1985

AN ALGEBRAIC APPROACH TO KNOWLEDGE STRUCTURING

HÀ HOÀNG HỐP

Hanoi State's Committee for Science and Technology
Hanoi - VIETNAM

ABSTRACT An algebraic approach to knowledge structuring is proposed which aims to show some preliminary aspects of knowledge representation, a so-called "generalized AND/OR graph" is defined and then combined to the structure of a such knowledge.

KEYWORDS Knowledge representation, "generalized AND/OR graph."

1. INTRODUCTION

Firstly, we present a knowledge algebra, then define a "generalized AND/OR graph". Some propositions and theorems are shown which affirm the admissibility of the construction. Our approach is based on some algebraic and analytical tools over a vector space which clarifies the combination of concepts from logics, relational structures and measures.

2. BASIS CONCEPTS AND DEFINITIONS [2], [4], [5], [6]

2.1. Definition (pseudo-metric space). Let S be a finite set of definite objects from realworld, and let F be a finite set of functions f_q $(1 \leq q \leq n)$ such that

The author receives one-year's fellowship from the Ministry of Culture and Education of Hungarian People's Republic and work at BME-HEI under the scientific direction by Dr. Gåbor Németh - Head of Comp. Tech. Dept at HEI.

(i) f_q: $S \times S \to \mathbb{R}^+$

where $S \times S$ is the Cartesian product of S and $\mathbb{R}^+$ is the set of all non negative real numbers.

(ii) f_q satisfies the following conditions.

(a) $\forall s_1, s_2 \in S$ $f_q(s_1,s_2) = f_q(s_2,s_1)$

(b) $\forall s \in S$ $f_q(s,s) = 0$

S with F, denote{(S,F)}, are named the pseudometric spaces f_q is called the distance function.

2.2. *Definition* (pseudodistance matrix). Call a matrix

$D = (d^a_{ij})$ $1 \leq i,j \leq k$ where

$d^a_{ij} = f_q(s_i,s_j)$ $\forall s_i,s_j \in S$, $1 \leq q \leq n$.

a pseudodistance matrix.

2.3. *Definition* (pseudo-Euclidean space). Call a vector space V a pseudo-Euclidean space if there exists in V an inner product <.,.> satisfying the following conditions

(i) $<u_1+u_2,v> = <u_1,v>+<u_1,v>$ $\forall u_1,u_2,v \in V$

(ii) $<cu_1,v> = c<u_1,v>$ $\forall u_1 \in V$, $c \in \mathbb{R}$

(iii) $<u_1,v> = <v,u_1>$ $\forall u_1,v \in V$

(iv) $<u_1,v> = 0$, $\forall u_1 \in V \Rightarrow v = o$

2.4. Definition (vector representation). Let $V_1, V_2, \ldots, V_p$ be the p pseudo-Euclidean subspaces in V such that with arbitrary i,j that $i \neq j$, $V_i \cap V_j = \{0\}$ and $V = \bigcup_{t=1}^{p} V_t$ Define a mapping g such that

$$g: \{(S,F)\} \to \{V_1, \ldots, V_p\} | g(S, f_q) = V_p\}$$

Call such g a vector representation of $\{(S,F)\}$.

2.5. Definition (equivalence relation). Let r be an arbitrary relation in V, r is called an equivalence relation in V if the following conditions are held

(i) $\forall v \in V, \quad v \; r \; v$

(ii) $\forall v_1, v_2 \in V, \quad v_1 \; r \; v_2 \Rightarrow v_2 \; r \; v_1$

(iii) $v_1, v_2, v_3 \in V, \quad v_1 \; r \; v_2 \;\&\; v_2 \; r \; v_3 \Rightarrow v_3 \; r \; v_1$

2.6. Proposition. $\{(S,F)\}$ generates a set of pseudo distance matrices.

2.7. Proposition. The set of all pseudodistance matrices forms a vector space with the usual operations on matrices. Consider a family of all subsets of the p-ary Cartesian product of V, denote $\mathcal{R}$.

2.8. Proposition. $\mathcal{R}$ is a relational algebra.

2.9. A formal procedure for knowledge structuring.

Step 1. Choose functions f_q $(1 \leq q \leq n)$ (heuristically) then define all pseudodistance matrices.

Step 2. Determine the function g.

<u>Step 3.</u> Define a problem-oriented relational algebra on basis of V.

<u>Step 4.</u> Determine an available equivalence relation r.

<u>Step 5.</u> Classify all elements of V into equivalence classes. Actually, there exists a natural quasi-order in $\prod_{\ell=1}^{p} v_\ell$, we return to theorem of Birkhoff that in our scopes, which has some interesting meanings.

2.10. *Theorem.* (Birkhoff, 1937). For $\prod_{\ell=1}^{p} v_\ell$ the relation

$$(v_1,\ldots,v_p)\ Q(v_1{}',\ldots,v'{}_p) \quad \text{if} \quad ((v_1',\ldots,v_p') \in r \Rightarrow$$

$$(v_1,\ldots,v_p) \in r, \quad r \in \prod_{\ell=1}^{p} v_\ell$$

defines an one-to-one mapping s from R to the family of all quasi-orders Q on $\prod_{\ell=1}^{p} v_\ell$.

2.11. *Definition* (knowledge algebra). A knowledge algebra is the following triplet

$$A = <\{(S,F)\},\ V,\ R>.$$

where {(S,F)} pseudometric spaces

V - pseudo-Euclidean space

R - relational algebra on basis of V.

2.12. *Definition* (knowledge). Knowledge is an equivalence class generated by an equivalence relation over V.

3. GENERALIZED AND/OR GRAPH

AND/OR graphs play a key role in the area of problem solving which is based on the first order predicate logics (see 8). We propose here a new concept which theoretically generalize the class of AND/OR graphs.

3.1. A schemata of generalization

There are 3 types of conjunctive AND.
There are 3 types of disjunctive OR.

(A1). ill. AND. Let A, B two arbitrary sentences $A \overset{ill}{\wedge} B$ if A,B happen within an interval of time by two forms.

- A happens before B, denote A b B,
- A happens after B, denote Aa B,

We put the estimation table for $A \overset{ill}{\wedge} B$ as follows:

	B	t_1			t_2
	A	a_o	a_1	b_o	b_1
t_1	b_o	ε_1	v_1	0	0
	b_1	v_2	ε_2	0	1
	a_o	0	0	ε_3	v_3
	a_1	0	1	v_4	ε_4
t_2					

where a_o - (for A) means that A is false and happens after B.

a_1 - (for A) means that A is true and happens after B.

b_o - (for A) means that A is false and happens before B.

b_1 - (for A) means that A is true and happens before B.

The same explains are applied to B.

$v_1, v_2, v_3, v_4 \in [0,1]$ and depend on the semantic of A and B.

$0 \leq \varepsilon_1 \ll 1$ $\qquad$ $0 \leq \varepsilon_3 \ll 1$

$0 < \varepsilon_2 \leq 1$ $\qquad$ $0 < \varepsilon_4 \leq 1$

(A2) AND. It is understood as the usual meanings.

(A3). well-AND; A, B happen within the time interval $[t_1', t_2'] \supset [t_1, t_2]$ by two forms similar to definition of ill-AND. The estimation table is as belows

		t'_1				t'_2
	B A	a_o	a_1	b_o	b_1	
t'_1	b_o	0	v'_1	0	0	
	b_1	v'_2	1	0	1	
	a_o	0	0	0	v'_3	
	a_1	0	1	v'_4	1	
t'_2						

$v'_1 > v_1$, $v'_2 > v_1$, $v'_3 > v_3$, $v'_4 > v_4$ and they depend on the semantic of A and B.

(01) ill.-OR. With the same forms of happening like above, the estimation table of A $\overset{\text{ill}}{\wedge}$ B is

	B	t_1			t_2
A		a_o	a_1	b_o	b_1
t_1	b_o	γ_1	u_1	0	1
	b_1	u_2	γ_2	1	1
	a_o	0	1	γ_3	u_3
	a_1	1	1	u_4	γ_4
t_2					

$0 \leq \gamma_1 \ll 1; \quad 0 \leq \gamma_3 \ll 1$

$0 < \gamma_2 \leq 1; \quad 0 < \gamma_4 \leq 1$

$u_1, u_2, u_3, u_A \in [0,1]$

u_i $(i=1,\ldots,4)$ depend on the semantic of A, B.

(O2). It is understood as the usual meaning.

(O3) well-OR. The estimation table of A well B is as follows.

	B	t'_1			t'_2
A		a_o	a_1	b_o	b_1
t_1					
	b_o	0	u'_1	0	1
	b_1	u'_2	1	1	1
	a_o	0	1	0	u'_3
t'_2	a_1	1	1	u'_4	1

$u'_i \quad u_1, \quad u'_2 \quad u_2, \quad u'_3 \quad u_3, \quad u'_4 \quad u_4, \quad u'_1, u'_2, u'_3, u'_4 \in [0,1]$

3.2. Definition (generalized AND/OR graph).

A random graph is named "generalized AND/OR graph" if the following conditions are fulfilled:

(i) its nodes are pairs of (sentence, time points),

(ii) its edges are one of the below operations
ill-AND,AND, well-AND
ill-OR, OR, well-OR.

3.3. Theorem. The set of all generalized AND/OR graph is isomorphic to

$$A_t = <\{(S,F)\}, V_t, R_t>$$

where (S,F) - pseudometric spaces.

V_t - a pseudo-Euclidean space for wchich it is combined by two spaces, a I-dimensional Euclidean space for the time-like vectors, and a(n-1)-dimensional Euclidean space for the feature-vectors,

R_t - a relational algebra over V, in which each relation is associated to a time point.

4. CONCLUSION

In this paper, our limited purpose is a firstly to propose a structure of knowledge. The results in this paper are the first steps for us to reach further ones in another paper later.

5, ACKNOWLEDGEMENTS

We would like to thank Dr. G. Németh for the scientific and administrative helps during the time we stay at HEI.

REFERENCES

[1] Cohn, P.M., Universal Algebra, New York, 1965.

[2] Dieudonné, J., Fondements d'analyse mathématiques, Dunod, Paris, 1960.

[3] Diday, E., Simon. J.C., Clustering Analysis, in Comm. and Cybers, 10, Ed. by K.S. Fu. Springer-Verlag, New York, 1980, pp. 47-92.

[4] Goldfab. L., An Outline of a New Approach to Pattern Recognition, in Control and Computers, vol. 13, Nr. 2, 1985.

[5] Grenander, U., Lectures on Pattern Theory, 3 volumes, Springer-Verlag, Berlin, 1978.

[6] Ha Hoang Hop, Truong Cong Dung, Algebraic aspects of matching problem (to appear).

[7] Laurière, J.L., Reprêsentation et utilisation des connaissances. Techniques et Sciences Informatiques, vol. 1. no. 1. 1982.

[8] Nillson, N.J., Principles of Artificial Intelligence, Springer-Verlag, Berlin, 1981 (first Edition).

[9] Rosenfeld, A., Fuzzy graphs, in L.A. Zadeh "Fuzzy Sets and their Applications to Cognitive and Decision Problems" Acad. Press, 1975.

The CDL Programming Support Environment from Dresden

Manuel Joiko

INTRODUCTION

At the implementation of system software or so-called "large software" for every kind of computer there is to take care of producing correct, adaptable, portable, and efficient programs. This software quality is influenced by the methods applied in its design, by the language chosen for its construction, and by the tools used.

Producing large software qualitatively differs from producing small software. The main problem is to overcome the high complexity of this kind of software. Programming-in-the-large that means also organization of an ordered collaboration between all which are engaged in manipulating the product all over the software-lifecycle. The production of system software is similar to the production of other technical products. According to this project planning and project management play an important role. There is to consider that the traditional programming languages including the connected tools were developed for the programming-in-the-small.

To produce correct software it is suitable to use a language which supports structured programming and stepwise refinement technics. Furthermore the language should give possibilities to specify the properties of data and algorithms. Analogous to algebraical specifications in numerical mathematics the specifications for system software should be described by metalinguistical means because of the nonnumerical nature of those programs. The programming system should include user-friendly error diagnostics based on the specifications. An interactive debugging tool using the object names of the source program is indispensable to prove the correctness. Furthermore desirable is the automatic generation of a debugging environment for incomplete programs. For the administration of large software a database-manager is a very powerful tool also.

Adaptable programs should be primary readable. The ideal should be achieved if there is only one language level handled all over the software-life-cycle and the program is the documentation itself. An user-friendly listing generator including pretty-printer, cross-referencer, and so on should support the readability. The used language should have a module and a library concept. A language oriented editor and an interactive intermodular analyzer including segmentation support should be available to perform the program modifications.

A high degree of portability can be achieved only by using a high-level language including the possibilities of variable target code generation.

Finally, the current development level of the most available personal computers demands for efficient programs in run-time and memory space. Therefore we still need optimizers on the source-/ intermediate-code level as well as on the target-code one. To have the possibility of using all machine properties is also indispensible for producing efficient code.

According to these claims a group at the Technical University of Dresden in 1974 started with the development of a programming support environment based on the system implementation language CDL proposed by KOSTER in 1971 /2/.

THE SYSTEM IMPLEMENTATION LANGUAGE CDL

The main idea of KOSTER by defining CDL was the investigation of Affix Grammars considering it as a programming language. Our CDL implementation realizes five important concepts.

The central role plays the <u>procedure concept</u>. A CDL program consists of definitions of several procedures (actions and predicates). Their functions are defined by the calls of other procedures. In this way the method of stepwise refinement on a problem oriented level is distinguished supported. On each refinement level the problem to be solved is named in the way of linguistic abstraction. The integrated decomposing mechanism supports not only the methods of structured programming, but enforces syntactically tree-structured programs.

Because it is impossible to continue the description of a procedure by calling of other procedures endless, CDL has the macro concept on the lowest refinement level to describe the base algorithms. These macros are defined by using the means of the target machine. In this way all properties of the target language can be used by CDL programming. For each project it is possible to define its own base machine. In this way CDL is an open-ended language.

Furthermore CDL has a module concept which permits to divide a CDL program into one main module and several attached modules. Each module is separately compiled and contains an exact interface specification with the import- and export-connections to other modules.

The data concept of CDL is very simple. It allows only to define data by determining the name, the memory space needed and the scope. On the other side CDL supports distinguished the definition of abstract data. The properties of these data are defined only by the operations, which are executed over them.

Finally, the library concept of CDL allows the collection of all definitions of base algorithms and constants in one module. These objects can be used in each other module without specifying or defining them once more. So the library concept permits the division of problem oriented and machine oriented constructs. If macros are only defined in the library, then it is possible to work in the modules only in a problem oriented way.

THE CDL PROGRAMMING SUPPORT ENVIRONMENT

In the following part the components of the programming support environment are shortly described. The interaction medium of all these tools is a virtual data memory of 128 k bytes.

1. Language Oriented Editor (LED)

With the editor it is possible to enter and modify CDL source textes syntax oriented in an interactive way. The editor handles the textes in an internal manner (EDD).

2. Modular Analyzer (ANA)

To this component belong the lexical and syntactical analyzers, the intermediate code synthesizer, and the semantic checker. It produces an intermediate program IP in the virtual memory. This intermediate program is a list structure which is very suitable for optimizations.
These tools are surrounded by a pretty-printer PPR) and a cross-referencer (CRF).

3. Intermediate Code Optimizer (ICO)

The intermediate code optimizer performs only machine-independent optimizations. It analyzes the intermediate program in the virtual memory and restores the optimized program back. The optimizations which are performed are described now:

- Procedures which are not reachable from the roots of the program graph are deleted.
- Data which are defined but never used will be optionally deleted.
- Procedures which are called statically only a few times, are removed by copying the procedure's body instead of the call.
- Are there some equivalent ends of alternatives, they are united and will be represented in the target program only once.
- Direct right recursive procedure calls are replaced by jumps to the start of the procedure.
- In case of equal actual parameters for one formal parameter in every call of a procedure, the parameter-list is decreased by substituting this formal parameter by the common actual parameter.
- To save memory space it is advantageous to remove some calls by internal parameterless procedures.

4. Variable Target Code Generator (TCG)

The expense of service and further development of a programming system is very high if there is a special code generator on every host machine for each target language and operating system. The experience showed that large parts of code generation are independent from the target language too. To standardize the code generators all machine-dependent algorithms were put away from the generator. So the remaining part is machine-independent and can be considered as basic operations of a programmable automaton. In a program for this automaton it is possible to represent those generation parts which are influenced by the target language. The code generator becomes an interpreter of those special programs. By writing different special programs any target code is producable by only one code generator.

To get an effective code generation on the one hand and also an acceptable comfort of writing such programs on the other hand, it was created a special Target Code Generation Language (TGL) to formulate the programs. A simple TGL-compiler (TGLC) translates those programs into an internal form (TGD) which is suitable for an effective code generation.

5. Variable Target Code Optimization (TCO)

An inspection of the target code showed that programs which are optimized in a machine-independent way only contain some typical unefficiencies. A good compiler-system should avoid such point of criticism by peephole optimizations on the target code level. The idea of a portable target code optimizer is similar to the variable code generation. The code-independent algorithms are summarized to the target code optimizer. The dependencies from target code are described in a special Transformational Language (TRL). This language bases on the formalism of attributed transformational grammars.

A program of this language describes the syntactical structure of the target code and some semantical aspects which are important for the optimizations. Furthermore it containsthe patterns for optimization conditions and the transformations. The optimizer works like an interpreter over the internal representation of a TRL-programm (TRD) produced by the TRL-compiler (TRLC).

6. Interactive Debugging Tool (CDT)

The variable target code generator of the CDL-system is a well-suited invention for interactive debugging. By a modified generation description - a new TGL-program - it is possible to generate a target code which contains all informations necessary for interactive debugging on a CDL source level. The CDT provides a lot of functions:

- setting trace- and breakpoints,
- setting conditions which can be connected with other functions,
- inspection of all active procedures for example to continue with any such procedure you like,
- showing and changing of any data you want,
- starting the debugging aid of the operating system for debugging on the target level,
- collecting statistical data about run-time,
- interrupt possibilities during the test by the operator,
- pocket calculator functions,
- showing CDL sources during the test.

Furthermore CDT provides 6 data types, hard copies, file service and handles errors which normally would cancel the program. The CDT is written in CDL and so highly portable.

7. Intermodular Analyzer (IMA)

The Intermodular Analyzer is an interactive system which analyzes the relations between the modules of a CDL-program. It gives a lot of informations about the whole program as a super-cross-referencer. Furthermore IMA supports the segmentation of CDL-programs and the construction of test frames.

8. Database Management System (DMS)

The Database Management System controls and manages the development of a CDL-program. It handles the several representations of the modules. The aim is that users only work in the CDL programming support and do not need no other environment. It prevents user operations from leaving a project in an inconsistent state.

9. Library Compiler (LIBC)

The Library Compiler transforms CDL-libraries into a special internal representation called library description LBD.

The pretty-printer and the cross-referencer process also libraries.

AVAILABILITY

In 1974 our group started with the implementation on the HONEYWELL-like R4000 (an older computer produced in the G.D.R.) The first version was programmed in FORTRAN. After that the programming system was written in CDL itself and was bootstrapped to other machines.

Presently the CDL programming support environment works on PDP11-like machines under RSX11 and on IBM370-like machines under OS. Now the group is engaged in completing the system on the several machines.

Members of our group started to bootstrap the system to INTEL8086-like and Z80-like machines under CP/M. Table 1 shows the availability of the components.

	PDP11	IBM370	IN8086	Z80	R4000
LED					
ANA	*	*			*
PPR	*	*			*
CRF	*	*			*
ICO	*	*			*
TCG	*	*			*
TCO	*				
CDT	*		*		
IMA					
DMS					
LIBC	*	*			*
TGLC	*	*		*	*
TRLC	*				

Table 1: Availability of the CDL-System (April 1986)

Every CDL-system can generate code for each other.

EXPERIENCES

Our CDL programming support environment proved at the implementation of several large software systems. Examples are a COBOL- and a BASIC-programming system, some compilers, interpreters, editors, and so on.

The experiences which are made show that

- the realization of large and powerful software systems is possible in a short time,
- the software implemented by our system is highly efficient and satisfies also commercial pretensions,
- the software implemented using the CDL technology is portable in a high degree.

REFERENCES

/1/ Kimm, R.; W. Koch; W. Simonsmeier; F. Tontsch:
Einfuehrung in Software Engineering.
Walter de Gruyter, Berlin, 1979.

/2/ Koster, C.H.A.:
Using the CDL Compiler Compiler.
in: Lecture Notes in Computer Science, vol. 21,
Springer-Verlag, Berlin, 1974.

/3/ Otter, W.; H. Loeper; T. Haenel; M. Joiko:
Programmentwicklung mit CDL1600.
Schriftenreihe Informationsverarbeitung,
Verlag Die Wirtschaft, Berlin, 1984.

MICROCOMPUTER-BASED SPECIAL MEDICAL INFORMATION SYSTEMS

P. Kerékfy, A. Kiss, I. Ratkó, M. Ruda

Computer and Automation Institute

Hungarian Acedemy of Sciences

Budapest

The authors and their colleauges have been engaged in developing health care systems since 1972. The paper reports on the authors' latest works on microcomputers.

Since microcomputers appeared, utilization of computers increases very rapidly.

Four special tasks are discussed below to illustrate the possibilities afforded by the cheapest personal computers.

1. The **infarction register** contains data of the South-East Budapest area, the population of which is above half a million. In the period of a year approximatly 3000 cases with acute myocardial infarction, or the suspicion of that, are registered. In more detail see in [1].

2. In the **register for extracorporal operations** the queue of patients waiting for heart operation (length about 1000 patients) is maintained by the system and it schedules patients for operation. After operation, data are stored in archive files. Statistics are collected to promote scientific research. In more detail see in [2].

3. A very large amount of data is stored in the **IHD-register** (Ishemic Heart Diseases) that is overlapping with the extracorporal heart operations register. This register contains data of preoperation tests such as: haemodynamics, x-ray, etc., operation events, post-operation follow-up.

4. The so-called **RGPC-register** (Richter Gedeon Pharmaceutical Company) contains data of Hungary, approximatly 2500 cases with (special) acute myocardial infarction.

The aim of the examination are: a) the appreciation of efficiency of the treatment with medicines **Tobanum** and **Rabenid** in the secondary prevention of acute myocardial infarction b) examination of side effects.

The study is based on fixed sample statistical methods. The method of the examination is double-blind, multicentral, random experiment controlled with **Placebo**.

One record of the register-file have 1064 data-items, the length of the record is 4614 characters.

Until now we have used five kinds of data forms:

(a) form of Basic Medical Examination

(b) two kinds of form of Following-up Medical Examinaton

(c) Event-form

(d) Death-form

The following-up examinations occur in the 1-st, 2-nd, 3-rd, 4-th, 5-th, 6-th, 8-th, 10-th and 12-th month after basic medical examinations.

On monitoring of the side-effects see [3].

The registers are realised on small microcomputers:

- eight-bit processor (Z80)
- 2 x 250 Kbyte floppy disk
- 64 Kbyte RAM (no ROM)

Our data management system developed for medical applications on small microcomputers is controlled by a monitor: micro-SHIVA. It is portable to any computer equipped with CP/M operating system, Z80 processor and 16 Kbyte RAM. The heart of this system is an extended full-screen editor. In more detail see [4].

Data manegement

The data manegement package provides the user program with functions needed in data management, and offers the user the opportunity of activating simple functions by function keys. These latter functions are the following:

- data input and data modification,
- record deletion,
- query by keys,
- display of record contents.

When the user pressed a function key the system waits for confirmation, and executes the command.

Data storage and modification

Database structure is defined by filling up a special form. It describes the record structure (names and attributes of data) and the search keys.

Data record consist of compressed fixed-length and variable-length data. Numeric data are stored as bit strings and are not aligned on byte boundary. Some data fields are assigned as keys. It is not required that a record be unambigously defined by its keys.

Data fields of a form displayed on the screen can be filled up at will, except those of R/O attribute. Names that may have been assigned to data fields of the form (while editing it) connect the fields to fields in database record. Contents of named fields are copied into the corresponding database record.

Searching

Two basic methods of query are used: one for immediate service of the user watching the screen, the other is intended to be used in batch-like processing. The first mentioned fast retrieval can be formulated in terms of keys allowing search for undefined key values and usage of partially defined keys, too. The key values are given by filling up a form. Searching is established by the use of a key-table that contains keys of each record. The key-table may contain unchanged or coded key values. In the latter case it is not required that the key values be decodable.

The other method is more flexible. In this case a special query form is filled up. Complex queries can be formulated defining conditions that consist of value list or interval list for data items. From these conditions complex formulae can be built up by the use of logical AND, OR and NOT. This method makes use of the key-table, too.

References

[1] I.Ratkó, M.Csukás: A data base management system for patients suffering from acute myocardial infarction, in: R.Trappl(ed.) Progress in Cyberenetics and System Research, Vol IX. Hemisphere, Washington, 497-501, 1980.

[2] I.Ratkó, M.Csukás, P.Vaszary: Computer registraton of patients waiting for cardiac operation, in: R.Trappl(ed.), Cybernetics and Systems Research, North-Holland, Amsterdam, 651-653, 1982.

[3] M.Csukás, E.Farkas, A,Krámli, G.Maróti, J.Soltész: Microcomputer monitoring of the side effects in Hungarian pharmacological study, In this volume.

[4] P.Kerékfy, M.Ruda: Form management by micro-SHIVA, In this volume.

Form Management by micro-SHIVA

Pál Kerékfy, Mihály Ruda

Computer and Automation Institute
Hungarian Academy of Sciences
Budapest, Hungary

Introduction

Nowadays it is a widely acknowledged fact that the conditions of the widespread efficient application of computers are ever less secured by the recent truly spectacular hardware developments themselves. The conventional technique of software development, the problem solving in various programming languages of lower or higher level, does no more satisfy the users, especially the growing number of the "personal" users, directly connected with the computers (with microcomputers or intelligent terminals), that is provided with very effective means as to their capabilities. The role of program packages performing tasks prepared in advance and exactly circumscribed has no decisive importance considering the total utilization of computer science. We can say with some exaggeration that of the "closed" programs only the game programs can expect undivided success.

It has become common knowledge in the field of informatics that such software means are needed which enable both the "lay user" and professional system programmer to generate systems satisfying their respective special demands in every respect. Two following points of view have to be stressed. One is to secure for the solving of any type of task the program generating tool operating with the most commonplace concepts, likely to exempt the user from programming work, even from the use of highest-level programming languages. The other is to provide the developer with a tool enabling him to satisfy the user's special demands, -- these demands can even involve the apparent breaking of the available computer capacity (this is possible only by disengagement from conservative means and by the application of new methods augmenting efficiency several times). In order to attain this goal the opennes of the program-genarating tool referred to in the first point of view must be secured(i.e. its ability to be completed by new elements, maybe to be broken down to parts).

The authors had already been investigating these issues before the start of the microcomputer flood, at the time of the appearance of the concept of software crisis, also in the field of conventional applications (mainframe computers with batch mode of operation) [5,6]. Embryos of these ideas emerged already in systems developed during the early '70s and introduced later in routine utilization [2,4,16]. These system developments raised besides practical problems of informatics theoretical issues in mathematics, too [17].

In the present paper our considerations outlined above will be expounded in the course of the presentation of a concrete microcomputer development. The presented means is the micro-SHIVA form and data management system, more exactly its layer next to the user, the form management one serving for input-output surface in the data management system. Instead of discussing the theoretical issues, we try to present our concepts with presenting this subsystem.

A short description of micro-SHIVA

The micro-SHIVA system handles forms similar to printed questionnaries, data sheets, files known to everybody on the screen of the computer. The shape of the forms, texts and fields printed on the same can be defined sitting in front of the screen with the aid of an editor. Later, the data, too, are read in on the screen. This is almost the same as the usage of a printed form - its pages can be turned over, some columns may be left out and data inserted in an arbitrary order. There are too important differences: there can be written only in the marked places (data fields) and errors committed in the course of insertion can be simply corrected. Forms filled out can be printed and data stored for subsequent searches and computations.

The system has been developed by the Computer Science Department of the Computer and Automation Institute and has already many applications.

By creating the micro-SHIVA form managment system we intended to provide both users and programmers of application systems with an easily applicable and efficient tool.

From the point of view of the user we considered essential that with the organization introducing the system the changeover to the computer should involve as little inconvenience and change as possible. At any rate we had to reckon with the fact that at most places there had already been developed well-rounded administrative systems. These frequently require the data-supplier to fill out forms or files. The data thus recorded will generally be thereafter carried over to punched card or magnetic media and processed by computer. The data handling on the screen changes this only as much as the data appear first on the screen instead of on the paper. As early as during the filling out certain verification can be performed and the attention of the person who fills in can be drawn on the errors which can thus be corrected without extra expenditure. After the filling in all known advantages of magnetic data recording are at our disposal (quick and exact information supply, use of network, avoidance of copying errors, etc). Later we shall see in the examples how flexible the format of printed forms can be and thus every kind of already existing print (based on official rule or custom) can invariably be utilized. All this can be attained by the micro-SHIVA user in a completely interactive way: the form for data entry, the checking conditions, the printing figure can be modified at any time, even during data entry.

The most important means for the designers of user systems is the form editor. This constructs the forms which the user will fill out. The location, length, considerations relevant to their control of the particular rubrics (fields) can be determined. Likewise their display and typography can be given.

The scope of utilization of the micro-SHIVA form managment system is very broad. It is enough only to reckon in how many walks of life various kinds of prints, viz. questionnaire, data sheet, invoice, receipt, order, bill of lading, application, decision and alike are being filled out. The data handling can be applied everywhere where data can be collected by filling out forms, files where data supplied by enumeration or tables have to be stored or calculations should be made with them.

Its typical field of application is the automated office (e.g. secretary station). It can be used for data survey of persons or objects (letters, books, engineering desings, items of inventory), their recording and printing, for the retrievals of files and registries. Letters, minutes, resolutions of arbitrary form or content can be printed on the basis of the data recorded.

In conformity with the demands and possibilities of the fields of application we have developed the micro-SHIVA data management system so it can serve as an efficient tool for the cheap desk-top computers, we are anxious to propagate it first to these -- systems already operating have also been implemented in such. The requirement of the present system is a Z-80-based microcomputer with floppy disk, printer and CP/M compatible operating system. Depending on the size of the task an about 15-30 kilobyte memory is necessary, thus additional user software carrying out special computational and other activities can run on the computer at the same time. Systems constructed until now operate on the microcomputers Varyter and Syster developed by the Computer and Automation Institute and on the microcomputers MOD-81 and MOD-81-M manufactured by MEDICOR Works.

The forms

The connection between the user working with the micro-SHIVA system and the programs is based first on forms. These forms appear on the screen and serve for data input and output, as well as, parametering of instructions.

The form consists of pages, of them only one at a time appears always on the screen, the forward and backward paging is activated by pressing a key. The selected sheet on the screen appears in the place (window) marked for the form. Its size can exceed that of the window both in horizontal and vertical direction.

The form (similarly to the printed ones) contains textual parts and fields (columns) to be filled out, fixed beforehand. The properties of the fields can be defined in the course of form editing and can be modified also in the program controlling the filling out of the form. The type of the field determines the basic points of view relative of its control (numerical, alphanumerical, etc.) and the mode of utilization (whether the user can only read or also write the same). A descriptive part of arbitrary length, not visible on the screen, can be attached also to the fields. This can be used e.g. for the definition of the connection with various database management systems.

As the possibilities for displaying forms offered by screens on the one hand and printers on the other hand differ form each other, the instructions controlling the printer can be given on the forms of the micro-SHIVA. These instructions do not come in view on the screen during the filling out of the form, they do not interfere with the display there. The control of the printer must be given independently from the just used printer in a uniform code whereafter the printing instructions translate the same into the language of the given printer. This possibility can be used to obtain sophisticated printed figures, e.g. tables, form prints printed in advance can be filled out.

The upper limit for the various quantitative characteristics concerning the forms is uniformly 255, this holds for the number of the pages, the number of lines and fields which can be placed in one page, the length of fields and lines.

Forms generated by the form editor are stored in floppy disks until further utilization. There are two ways of storage: either the area occupied on the floppy or the size of the memory required by the program during the filling out may be optimized.

The following figures show some forms, these are at present involved in routine applications. Fig. 1 presents a form for simple data recording as it just appears on the screen. Fig. 2 shows a form containing the same data and suitable to be printed out on the printed matter used earlier and then filled out by

[13] presents the various possibilities of the micro-SHIVA form management and an interesting example of application of the same.

Form editing

The form editing in micro-SHIVA is performed by a special text editor program. This program, similarly to the other softwares of this type -- but contrary to the style of other functions of the micro-SHIVA system (filling out forms, database handling) is a separate "closed" program in the conventional sense, capable of being handled independently of other functions of the micro-SHIVA. It secures to the user all possibilities the full-screen editors have. On the other hand, this program does not contain the high-level services of sophisticated text editors, e.g. automatic alignment of the right margin, block move operations, etc. This is due to the particularity of the form editor that it does not serve for writing of conventional texts, only for editing just forms.

Text editors are generally made to generate texts to be printed. The forms, however, frequently get not into the printer but into the screen. Thus the conventional text editing functions (proportional character arrangement, double-column makeup, etc.) become meaningless, at the same time demands resulting from the special characteristics of the forms come to the front: horizontal shifting of the edited text (for broad printed matter or text to be printed with high density, thus producing long lines on the screen), formation of varied written form (different letter sizes, text parts printed with various intensity). This problem necessarily emerges if we wish to fill out a printed form which -- altough not prepared for computer processing -- has already been widespread and accustomed. (Fortunately today these can be realized by cheap matrix printers.)

Compared with the conventional text editors the most important difference obviously is due to the fact that the form editor cares also for the location and specification of the data fields. Altough other text editors, too, perform such functions (WordStar), but by using them only as auxiliary features. On the other hand, the micro-SHIVA form system serves primarily the context-aided data flow and not the activation of parameters embedded in an edited text (e.g. addressees of a letter in MailMerge).

By specifying the data fields and with the aid of control functions put in the text the micro-SHIVA editor influences the behaviour of the program "operating" the edited form. The edited form can therefore be considered at the same time as a control program and the form editor as a program generator. Thus the program is an amalgam of the widespread text editor, spreadsheet and report generator tools. The diversity of utilization will be illustrated later by examples of application.

As to general points of view, the "visual technique" highlighted in the form handling should be stressed. For the principle that information imbedded in the form should be visible (e.g. the specification of the fields takes place in their own context) is valid not only in the stage of editing but the visibility of data flow is advantageous also in the course of the utilization of the form: the signalization of errors (blinking, textual error messages, sound signal), synchronous (occuring on one screen) display of coherent data grouped according to various criteria, e.g. repeated display of certain prominent data (heading) or multiple display of particular data in different data environment, etc.

We have considered th simplest possible realization of the form editing process also important. Thus the editing program is operated not by various menus but by using function keys. Besides only the following commands should be known: Q (quit), E (exit), T definition of tabulator positions. (One can enter in or return form the command mode only by pressing the enter key.)

Among the function buttons there are besides the usual cursor and screen mover as well as insert and delete keys, as a result of the speciality of the form editor, the following special functions: horizontal picture moving, definition of the location of data fields, specification of data fields, determination of the location of control characters, displaying or hiding them, graphic mode of operation switch.

The specification and modification of the fields comes about in the screen area separated from the edited text, by filling out fields for: 1. Type of the field (numerical, alphabetical, alphanumerical or arbitrary content), and within the same can it be written or only read, besides it can be "instantenous" type or not. This latter notion means that the input process of the data handling system regains the control at once at the time of reading in the data field after the entering of every character, otherwise this occurs only in case of the filling out the whole field, or at the time of the activation of some other special function (i.e. if we leave the field by moving the cursor). 2. Initial value. 3. The indication of the so-called interface which can result in the connection the field to a database field, or the starting of some other process. On the other hand, fields in the form are not neccesarily connected to those of a database.

Control characters get on the screen only in the stage of editing but even here they can be hided (by a function key) if we wish to see the form as it appears on the screen or printer during the filling out. The function of the control characters (character strings) is to generate processes independently of the data fields. This could become necessary for the editing of printed matter requiring variable letter type or for calculations needed in a report.

The graphic mode of operation enables the application of semi-graphic characters to which there are no keys assigned.

Filling out forms

This is the part of micro-SHIVA system with which both the "simple" user and the system programmer are brought into contact. Who inserts or interrogates data in a system built on micro-SHIVA does not practically do anything but filling out forms -- and his instructions, wishes are realized at once (if the system programmer did his work well). This latter has plenty of procedures and adjustable parameters at his disposal to build up a system to serve the user maximally.

The filling out of form, different from form editing (meaning the use of a closed program) involves the assembly and parametering of procedures to be found in libraries. In this sense no "form-filling program" exists, on the contrary there is a special one for every user and application.

The programmer can be brought into contact with the collection of form-filling procedures on different levels. He can commit nearly everything to the basic system (and in the majority of tasks this is completely suitable as a first solution). He can give the name of the form to be filled out or can cause it to be interrogated from the user, can write data form the program in the fields of the form, can read what the user has written, can store data in the database and let the form to be printed. To this effect he has to learn some (less than 10) procedures and call them in appropriate order. For the simplest case, viz. if the names of the data base and the form have to be asked from the user and the data are to be subjected to basic check only, he obtains a ready program, too.

On the other hand, the programmer is enabled to control particular procedures on a lower level. Thus he can modify the characteristics of data fields, or can display on the screen in several windows at the same time several forms, or different pages of the same form. The application program can easily gain control over every function handled in the basic case directly by micro-SHIVA (e.g. paging, check of fields, printing), and thus modify them as needed.

Without the claim of completeness we enumerate some solutions applied at present in systems operating routinely.

Examples of applicatons, conclusions

Medical applications. The elaboration of the micro-SHIVA concept started within the frame of our research tasks connected with medical information systems. The usability of the concept was justified initially on a mainframe computer model [15], later with sample systems fed into a microcomputer [7]. Thus the transfer of various medical information systems to simple 8-bit microcomputers came about [3,8,9]. The importance of this issue is due to the fact that especially in the beginning (in the late 70's and early 80's) but even now the role of the microcomputers and in particular of the small-power 8-bit computers in computer applications is debated. The usability of the medical information systems mentioned above has proved the utilization of the microcomputers. Here it should be noted that in determining the usability one cannot think in simplified notions: one must consider also quality of the peripherials (disk, printer, keyboard, screen) as a factor of basic importance.

When designing the microcomputer user systems and judging their efficiency the local networks, making their appearance at present in an ever growing number of places, have to be taken into account, too. A microcomputer network can serve as an efficient device in medical applications (especially in the case of integrated hospital systems [1,10]).

Figures 1 and 2 have been taken as examples from this field of application.

In medical applications, especially on a microcomputer network on the various points of which the data protection should be realized on different levels, the application of the micro-SHIVA system is very advantageous. In our network system based thereon double data protection is accomplished. On the one hand, there exists a protection available also in systems operating on one computer: only who has got a suitable form to this effect, can modify or have a look on the particular fields of the data base. On the other hand, the other level of the protection ensures the access of every working station to those data only which have been assigned to it by the station having the data (the network data flow, too, is carried out through forms).

<u>Application in management</u>. The micro-SHIVA developments have soon transcended the scope of medical applications. As set forth above, the micro-SHIVA system is for all practical purposes a general tool. Over and above the fast and comfortable handling of simple data files, it can also be applied to build up more sophisticated systems. Our first such system was a program system helping to determine old age pension [14]. This program system wants to replace a clumsy, by its nature slow mainframe system. It also wants to ensure possibilities (e.g. with the help of management administration) which cannot be realised in a batch system. Our third figure presents a form filled automatically as a result of the processing of several other data sheets filled out previously.

A new point of view of data security got prominence in this management application. While in medical applications first of all unauthorized access to data has to be prevented, in a system containing also financial calculations first unauthorized data modification should be made impossible, whereas the chief object of the system is just to insert as easily as possible the permanently completed and modified data in the course of management to the existing ones. This contradictory issue can easily be settled by the flexibility of the micro-SHIVA form handling and its sensitivity to the modification of data fields. The operator can print a valid document only if he did not touch the basic data after the run of the program generating the computed data. Besides, the validity of data derived from previous work phases is also examined by the partial systems working such data.

Industrial application. Management technique has its peculiarity in every field. This accounts for the substantial difference of user systems applicable in the particular areas, even if they stand on a common ground. This is true also for the medical and pension calculating systems presented above. Within the frame of industrial applications a pharmaceutical factory laboratory system and a machine industrial program package has been completed. We shall deal with this later system more in detail. This micro-SHIVA-based program package made for the GANZ-MÁVAG Railway Vehicle Factory [13] surpasses the systems enumerated until now that it carries out data flow by applying micro-SHIVA form-handling technique among data bases differing substantially from each other as to their structure. The task of this program system is to support, on the one hand, the preparation of the so-called corner-table getting on the draft of the particular components and finished products (enumertion of components and building blocks presented on the draft), and - as a pecularity of this application - by passing along the row of the hierarchically systemized drawings (produsts, building blocks, components) draw up the so-called work-sheets defining the production process errorless and quicly compared to the conventional solution (see Fig. 4). This work-sheet contains the specification of the components (building blocks) necessary to produce the particular products or building blocks, their quantities and quality indicators. Thus this program system automates and at the same time accelerates a lot of intellectual works which, because of the possibility of human error, are in their conventional form not only slow and expensive but can also do great harm in the production (wrongly scheduled production, lack of or surplus in components).

The paper [13] illustrates well by its many examples the possibilities of the application of the micro-SHIVA form-handling technique. At the same time it presents the operation of the micro-SHIVA-based user system by a concrete example.

The connection between form-handling and data bases.

The micro-SHIVA system contains besides the form handling subsystem also database-handling capabilities. The form-handling, however, can be built on other database-handling systems, too.

Such a possibility is examplified by a record system for corresspondance (letter filing) which uses as an input-output surface the micro-SHIVA form-handling, as a data base the data-handling subsystem of an inventory system (originally made for wholesale and retail trading companies).

As a final conclusion it can be said that the great number of user demands coming from various fields has justified the utility of the development during the relatively short space of time passed until now. The diversity of information materials processed, the largeness of their size show the efficiency of the microcomputer application. As examples we can point to one of our health systems (drug effect test) in which we process 15-20 Mbytes of data (by using floppy background store) or to the social security task in which -- although distributed to several workstations -- the handling of the data of about 100.000 persons should be performed.

References

1. Bakonyi P., Békéssy A., Demetrovics J., Kerékfy P., Ruda M., A Microcomputer-Network Based Decision Support System for Health-Care Organization, In: Gertler J., Keviczky L. (eds.), A Bridge Between Control Science and Technology, Pergamon Press, Oxford (1985) 1651-1658.

2. Csukás M., Greff L., Krámli A., Ruda M., An approach to the hospital morbidity data system development in Hungary, In: Colloques IRIA, Informatique Médicale, Vol I., IRIA, Toulouse (1975) 381-390.

3. Csukás M., Kerékfy P., Ratkó I., Ruda M., Microcomputer Based Cardiological Patient Registers, In: D.A.B. Lindberg, P.L. Reichertz (eds.), Lecture Notes in Medical Informatics, Vol. 24, Springer-Verlag, Berlin-Heidelberg-New York (1984) 240.

4. Csukás M., Krámli A., Ruda M., The computer realisation and first experiences of the hospital morbidity study, WHO Statisztikai Vándorszeminárium, Budapest (1974).

5. Kerékfy P., Krámli A., Ruda M., SIS79/GENERA Statistical Information System, In: R. Trappl et al. (eds.), Progress in Cyberbetics and Systems Research, Vol. XI, Hemisphere, Washington (1980) 123-128.

6. Kerékfy P., Ruda M., Program Optimization and Manipulation on User Level, In: R. Trappl (ed.), Cybernetics and Systems Research, North-Holland, Amsterdam (1982) 797-802.

7. Kerékfy P., Ruda M., A System Model for Microcomputers in Health-Care Applications, In: W. van Eimeren et al. (eds.), System Sience in Health Care, Springer-Verlag, Berlin-Heidelberg-New York (1984) 1419-1422.

8. Kerékfy P., Ruda M., Micro-SHIVA user friendly information system development in medical application, In: D.A.B. Lindberg, P.L. Reichertz (eds.), Lecture Notes in Medical Informatics, Vol. 24, Springer-Verlag, Berlin-Heidelberg-New York (1984) 235-239.

9. Kerékfy P., Ruda M., Distributed systems on simple microcomputer architectures, In: *Proceedings, IFIP 84 Symposium, Network in Office Automatization*, Sofia (1984) 444-447.

10. Krámli A., Ruda M., Csukás M., Galambos M. , Large sample size statistical information system for HwB, In: E. Diday et al. (eds.), *Data Analysis and Informatics*, North-Holland, Amsterdam (1980) 457-462.

11. Máté L., Ruda M., Microcomputer system for determining old-age pension, *PERSCOMP*, Sofia (1985) (microfiche).

12. Ratkó I., Csukás M., Vaszary P., Computer Registration of Patients Waiting for Cardiac Operation, In: R. Trappl (ed.), *Cybernetics and Systems Research*, North-Holland, Amsterdam (1982) 651-653.

13. Ruda M., Statistical Information System with Health Service Application, In: J. Gertler (ed.), *Fourth Winterschool of Visegrád on the Theory of Operating Systems*, *MTA SZTAKI Tanulmányok* 87(1978) 167-172.

KÓRLAPFEJ

1 Az adatgyüjtést az egészségügyi miniszter a 87 010 31 sz. alatt rendelte el.

1 Törzsszám: 0 1 0 0 0 0 3 1

2 Intézmény neve: ORSZ. [redacted]

3 Intézmény azonositója: [redacted]

A BETEG SZEMÉLYI ADATAI

4 Neve: Kis Borbála 40 éves

5 Leánykori neve:

6 Anyja neve: Kis Dezsőné

7 Születési helye: Sopron

8 Személyi száma: 2 4 6 0 1 0 3 8 7 7 9

9 Állandó lakása: Sopron Győri u.2. 8234 Területi kódja:

10 Ideiglenes lakása: Területi kódja:

11 Személyi ig. száma (KÜLFÖLDI esetén a KITÖLTÉSI UTASÍTÁS szerint)

12 Foglalkozása (nyugdíjas jellemző fogl.): eladó FEOR kódja:

13 Munkahelye: V. bolt

14 Gazdasági aktivitása: aktív (1) tanuló (2) öregségi nyugd. (3) rokkant nyugd. (4) htb (5) egyéb (6)

15 Családi állapota: nőtlen, hajadon (1) házas, élettárs (2) elvált (3) özvegy (4) — 1

16 Legközelebbi hozzátartozója (neve, címe, telefonja): Sopron Győri u.2. 8234 Kis Dezső

17 Iskolai végzettsége: 0 (1) 1–3 (2) 4–6 (3) 7–8 (4) szakmunk. (5) középfokú (6) felsőf. (7) ism. (8) — 2

18 Állampolgársága (csak KÜLFÖLDI esetén): KGST-ország (1) egyéb (2)

A FELVÉTEL KÖRÜLMÉNYEI

19 Beutalási diagnózis: Rossz a szive BNO kódja: Beutalás oka (■ szerint):

20 Felvétel időpontja: 19 8 6 0 6 2 8

21 Sürgősség: sürgős (1) nem sürgős (2) sürgős mentő (3) nem sürgős mentő (4)

22 Ágy rendeltetése: városi (1) megyei (2) regionális (3) országos (4)

23 Beküldő orvos neve: Dr Kis
– munkahelye: XV. rend.
orvosi szolg. száma:

24 NYILATKOZAT:
A kapott felvilágosítás alapján a házirendet kötelezően betartom. ______ a beteg aláírása
A kapott felvilágosítás alapján a gyógykezelésbe, orvosi beavatkozásba, műtétbe beleegyezem. ______ a beteg aláírása

2 AZ ÁPOLÁS ADATAI

25 OSZTÁLY	A FELVÉTEL időpontja év	hó	nap	óra	perc	oka ■	Ágy jell. ●	Diagn. tip. ▲	A BETEGSÉG MEGNEVEZÉSE	BNO kódja 1	2	3	4 5
0 1	86	06	28	11	12								

osztályos orvos aláírása

26

osztályos orvos aláírása

■ Beutalás és felvétel oka: hirtelen rosszullét (1) baleset, mérgezés (2) gyógykezelés (3) utókezelés (4) kivizsgálás (5) kontroll (6) szülés (7) kísérő (8) egyéb (9)

● Ágy jellege: felvételi (1) intenzív (2) őrző (3) általános (4) krónikus (5) szanatóriumi (6)

▲ Diagnózis típusa: átutaló (1) felvételi (2) ap. ind. fő (3) szövődmény (4) kísérő (5) klin. halálok (6) kórbonc. alap (7) kórbonc. halálok (8) kórbonc. kísérő (9) sérülések, mérgezések külső okai (E) nem betegség és nem sérülés, de eü. szolgáltatás (V)

27 Az első műtét időpontja: 19

28 A műtét WHO kódja:

29 A távozás vagy halál időpontja: 19 8 6 0 7 0 3

30 Kiírási állapot: síne moro (1) gyógyult (2) javult (3) változatlan (4) rosszabb (5) meghalt (6) egyéb (7) — 3

31 Továbbkezelés: 3. oszt. áth. (1) más korh. áth. (2) spec. szintre helv. (3) saját oszt. vissza (4) saját int. vissza (5) alapell. át. vissza (6) járóbet. szakr. átadva (7) kez. és megf. nem ig. (8) egyéb (9)

PH

kiállításért felelős aláírása

intézeti orvos aláírása

```
F E L V É T E L         Törzsszám:  00031
                                                   Oszt., ágysz.: /1

A beteg neve: Kis Borbála
Leányk. neve:
Anyja neve:   Kis Dezsőné
Szül. helye:  Sopron                  Személyi száma:  00000000000
                                          neme: 2  szül.: 460103  ANH: 8779

                                    Szülők adatai

            Anyja fogl.: eladó              (megh.:19 00 )    60 éves

              Apja neve: Kis Dezső
                  fogl.:                    (megh.:19 00 )    62 éves

                                    Házastárs adatai

                    neve: Nagy János            sz.sz.: 14012311119
          A házasságkötés helye:    Sopron     , ideje: 660404


Állandó lakóhelye: Sopron
    utca, házszám: Győri u.2.
     irányítószám: 8234     megye kódja: 00          területkód: 00000

Ideiglenes lakása:
    utca, házszám:
     irányítószám: 0000                              területkód: 00000

 egköz. hozzát.(gondv.)neve: Kis Dezső             0 1:apa, 2:anya, 3:házt.
                       címe: Sopron                0 1:áll. 2:ideigl. lakh.
              utca, házszám: Győri u.2.
               irányítószám: 8234                tel: 000000
```

SzÜV Nyomda 84.2461

[redacted]
Járműszerkesztés

Gyártmány: [redacted]

Leválogatás neve: [redacted] műhely tételei

DARABJEGYZÉK LEVÁLOGATÁS

Főköltséghely: 17
Darabjegyzékszám: 012
Lapszám: 001

AZ ALKATRÉSZ LEÍRÁSA	A tétel csládfája a fő összeállítási rajzból kiindulva	Gyártó üzem, tasakszám				DARAB/VÁLTOZAT									
		1.	2.	3.	4.	0.	1.	2.	3.	4.	5.	6.	7.	8.	9.
Név:Diesel motor Anyagminőség: Méret: 16PA4V-185VG 001 777/002	1113-511000/001 Hivatkozott rajz:		002 -AA- 1723 -04-		009 -BB- 1762 -05-	Nagyátfutású tétel jele:N Nyers méret: H. tev.szám:102-191 Igénylési szám: 1411 Dimenzió: db Kiemelt technológia: 1 1.00									
Név:Fögenerátor Anyagminőség: Méret: WD 630 bW8 ACW 100532 002 777/003	1113-511000/002 Hivatkozott rajz:		002 -AA- 1723 -04-		009 -BB- 1762 -05-	Nagyátfutású tétel jele:N Nyers méret: H. tev.szám:102-189 Igénylési szám: 1412 Dimenzió: db Kiemelt technológia: 1 1.00									
Név:Alsó fegyverzet Anyagminőség: Aö 400 Méret: M.SZ. 613504 003 777/005	1113-511000/003 1001-500510/001 1001-500600/001 Hivatkozott rajz:	010 -QQ- 1723 -09-	002 -AA- 1723 -04-		009 -BB- 1762 -05-	Nagyátfutású tétel jele:_ Nyers méret: H. tev.szám:108-171 Igénylési szám: 1241 Dimenzió: Kiemelt technológia: 6 6.00									
Név:Felsö fegyverzet Anyagminőség: A50 Méret: átm:220-104 004 777/006	1113-511000/003 1001-500510/002 1001-500600/002 Hivatkozott rajz:	010 -QQ- 1723 -09-	002 -AA- 1723 -04-		009 -BB- 1762 -05-	Nagyátfutású tétel jele:_ Nyers méret:átm. 220-200 H. tev.szám:106-171 Igénylési szám: 1242 Dimenzió: Kiemelt technológia: 6 6.00									
Név:Gumibetét Anyagminőség: gumi Méret: rajz szerint rendelve 005 777/007	1113-511000/003 1001-500510/003 Hivatkozott rajz:	010 -QQ- 1723 -09-			009 -BB- 1762 -05-	Nagyátfutású tétel jele:_ Nyers méret: H. tev.szám:124-171 Igénylési szám: 1407 Dimenzió: Kiemelt technológia: 6 6.00									
Név:ütközötest Anyagminőség: A 50 Méret: átm: 170-80 006 777/008	1113-511000/004 1113-511600/001 Hivatkozott rajz:	010 -QQ- 1723 -09-			009 -BB- 1762 -05-	Nagyátfutású tétel jele:_ Nyers méret:006-ról H. tev.szám:122-171 Igénylési szám: Dimenzió: Kiemelt technológia: 6 6.00									

An approach to implementing principles of discrete event simulation on PROLOG

by

Udo Konzack

Bereich Informationsverarbeitung
Sektion Mathematik
Humboldt-Universitaet zu Berlin
DDR - 1086 Berlin PO Box 1297
German Democratic Republic

Abstract

Since the early 80'th the programming language PROLOG has worldwide become of great interest with regard to the aim of developing new computer architectures. Especially the proclamation of the Japanese Fifth Generation Computer Project has given rise to a lot of investigations in the field of logic programming. But most of this research work is directed to AI applications. In the author's opinion it would also be worthwhile and nessesary to investigate the suitability of PROLOG for various other purposes differing from AI problems.
This contribution deals with some aspects of using PROLOG for solving problems of dicrete event simulation. The following explanations are based on experience made by using the micro PROLOG system on an 8 bit personal computer. PROSIT (PROLOG SIMULATION TOOLS), a prototype simulation system has been implemented and all examples and principles mentioned have been tested and executed on it. It should be mentioned that serious applications of course would require higher performance computers. But for studying principles the micro PROLOG system proves to be surprisingly powerful, which is mainly due to its simple LISP like syntax.

0. Introduction

It is presumed that models are regarded as systems of parallel processes. This gives rise to the following points of emphasis:
- process description/model representation in PROLOG
- process scheduling
- supervision of shared resources
- process communication
- random number generators
- evaluation and statistics
- user interface

On the other hand the question arises which new qualities for simulation can be achieved by using PROLOG, which provides the following special features:
- compatibility of data and programmes
- relational (nondeterministic) programming
- backtracking
- untyped variables
- unification

1. Model Representation

A process can be replicated on a PROLOG clause in a relatively natural way. The clause head would correspond to the identifier of the process and the tail to the sequence of its actions. An action can also be represented by clauses in the same way. This allows a systematical, hirarchical and easy to survey process description.
The problem is just that an ordinary PROLOG theorem prover can not run two or more processes in parallel but only sequentially one after the other. Moreover, global data which would be needed for providing message management, resource access control and process synchronization in time are not available.

```
((_consumer)(decide_for_product X)
            (_send ((_producer) (X needed)))
            (_receive ((_producer) (X available)))
            (_consume X)
            (_consumer))

     ((decide_for_product product_1) (draw 0.4))
     ((decide_for_product product_2)

     ((_consume product_1) (_hold X (randint 3 7 X))
     ((_consume product_2) (_hold X (randint 2 5 X))

((_producer)(_receive ((_consumer) (X needed)))
            (_produce X)
            (_send ((_consumer) (X available)))
            (_producer))
     ((_produce X) (_prepare X)
                   (_complete X))

           ((_prepare product_1) (_seize tool_1)
                                 (_hold 5)
                                 (_release tool_1))
           ((_prepare product_2) (_seize tool_2)
                                 (_hold 3)
                                 (_release tool_3))

           ((_complete       X ) (_seize tool_3)
                                 (_hold 7)
                                 (_release tool_3))
 ((tools (tool_1 tool_2 tool_3)))
```

Fig.1: model of a consumer producer problem

For this reason a modified PROLOG theorem prover which provides a coroutine mechanism and special built-in predicates for synchronization is needed. Such a theorem prover is implemented in PROSIT and Fig.1 shows a model of a "consumer producer problem" as it can be executed on it.

2. Process Scheduling

The scheduler of the PROSIT prototyping system is a special theorem prover completely implemented on PROLOG.
The processes which are active during a simulation experiment can be regarded as active PROLOG goals. Consequently the synchronization of processes has its equivalent in the synchronization of proving goals. But since in sequential PROLOG only one goal can be active at a certain time, the scheduler must be able to freeze up the incomplete prove trees of all suspended goals in such a way that after changing the active goal the prove can be continued from the position achieved last.
The scheduler is driven by specifically defined predicates (scheduling predicates) and if such a predicate is found during proving the active goal, it will be checked whether the active goal must be suspended or not.
At present PROSIT provides the following additional predicates:

<1> (_hold X) : suspends the active process for X time units

<2> (_hold X Y) : similar to <1>; Y specifies the rule for computing X

<3> (_send (X Y)) : the active process sends the message Y to process X

<4> (_receive (X Y)) : the active process waits for receiving the message Y from process X

<5> (_seize X) : the active process seizes resource X if X available, else passivate

<6> (_release X) : the active process releases the resource X

<7> (_available X Y) : if all resources occuring in X are available, Y ="yes", else Y ="no"

<8> (_wait X) : the active process becomes passive until fulfilment of condition X

<9> (_time X) : X gets the value of the current model time

<10> (_new X) : a new instance of process X becomes active at the current time

<11> (_new X at Y) : similar to <10>, but at time Y

<12> (_new X name Y) : similar to <10>, but with internal identifier Y

<13> (_new X name Y at Z) : similar to <11> + <12>

All these predicates are prefixed by the "_" sign for indicating dynamical model parts. Each functor of a model clause which contains a prefixed term must also be prefixed by a "_". This enables the theorem prover to minimize the effects of partially double-entry bookkeeping and slowing down of execution speed which is caused by the scheduling overhead, by deligating the prove of all nonprefixed subgoals to the built-in theorem prover which can do it much more efficiently.
The _new procedure serves for dynamical invoking of instances of predefined processes in an object oriented manner. Each process invoked is entered into a certain central queue. So for instance the SQS contains all processes with a known activation time and the process with the lowest one is the active process.

3. Resource Management

There is one central queue which contains all processes waiting for the release of a resource. The availability of a set of resources can be checked using the predicate _available. This allows e.g. to model the philosopher problem in the following deadlock free way, where the process waits in the condition queue until both resources are available.

```
((_philosopher X) (_think) (_eat X) (_philosopher))

       ((_think) (_hold X (randint 5 9 X)))

       ((_eat X) (my_forks X Y Z)
                 (_wait ((_available (Y Z) yes)))
                 (_seize Y)
                 (_seize Z)
                 (_hold x (normal 5 1 x))
                 (_release Y)
                 (_release Z))

           ((my_forks 1 fork1 fork2))
           ((my_forks 2 fork2 fork3))
           ((my_forks 3 fork3 fork4))
           ((my_forks 4 fork4 fork5))
           ((my_forks 5 fork5 fork1))

  ((forks (fork1 fork2 fork3 fork4 fork5)))
```

Fig.2: model of the philosopher problem

Resources must be defined as an argument list of a fact in the model file.

4. Process Communication

PROSIT contains two central lists for establishing process communication via a mailbox principle. One of these lists can be considered a mailbox and the other a queue of waiting receivers. A message can be an arbitrary PROLOG term - even a part of a PROLOG programme, too.

The receiver and sender need not be fully determined. By using variables the unification algorithm guarantees that the response reaches the right sender. This is illustrated by the following master slave example.

Other communication principles as e.g. rendezvous principle or broadcasting are also easy to implement.

```
((_master X)      .     .     .
                        .
                        .
             (_send    ((_slave Y) (execute task Z)))
             (_receive ((_slave Y) (task Z executed)))
                        .
                        .
                  .     .     .         )

((_slave X)  (_receive ((_master Y) (execute task Z)))
             (_execute Z)
             (_send    ((_master Y) (task Z executed)))
             (_slave X))

        ((_execute X) ¦X)
```

Fig.3: master slave principle

5. Random Number Generators, Statistics and Evaluation

Of course most PROLOG systems are not well-suited for numerical computation, especially if they only provide integer arithmetic. But also if there is a real arithmetic, built-in functions like EXP, LN, SIN, . . . are mostly missing.
For this reason PROSIT provides only equally distributed and normally distributed random number generators. Other distributions must be read from a file or calculated via linear interpolation of a density function.
Statistics and evaluation have not been supported yet.

6. Final Remarks

PROLOG proves to be appropriate for discrete event modelling and simulation. Model description can be done on a highly declarative level. One of the main advantages of PROLOG in contrast to other programming languages is the compatibility of data and programmes, which enables to implement arbitrary synchronization mechanisms in a relatively easy way. Due to the implementing on PROLOG it was very easy to modify and to transport the simulation system to other computers. Experience has shown that especially process communication principles can be modelled and carried out very conveniently on the basis of a PROLOG unification algorithm. Untyped variables are of great advantage in this case.
It can be expected that futural PROLOG systems will overcome many problems occuring at present e.g. with regard to the arithmetic or to the access to features of the operating system. Furthermore, execution speed and memory space will no longer be the bottle-neck when special PROLOG computers are available. Then a PROLOG simulation system can be a serious competitor against traditional systems.
Todays application of PROLOG for simulation could also be worthwhile when the system to be modelled consist of a small dynamic part compared to a large logical part e.g. if the model has to reflect principles of AI.

7. References

[1] Clocksin,W. F.;Mellish, C.: Programming in PROLOG,
Springer Verlag, Berlin 1980

[2] Kowalski,R.: Logic for Problem Solving,
North-Holland 1979

[3] Birtwistle,G. M.: Discrete Event Modelling in Simula,
The Macmillan Press Ltd., London 1979

[4] Futo,I.; Szeredi,J.: T-PROLOG User Manual,
Inst. for Coord. of Comp. Techn., Budapest 1984

[5] Futo,I.; Gergely,T.: A Logical Approach to Simulation
(TS-PROLOG)
Proc. of International Conf. on model Realism,
Bad-Honnef, FRG 1982

[6] Clarck,K. L.; McCabe,F. G.: micro-PROLOG: programming in
logic,
Prentice Hall International, Englewood Cliffs 1984

[7] Campbell,J. A.: implementations of PROLOG,
Ellis Horwood Limited, New York 1984

MICROCOMPUTERS AS ADAPTIVE TOOLS BETWEEN INVALIDS AND THEIR ENVIRONMENT

Juliusz L. Kulikowski
Inst. of Biocybernetics
and Biomedical Eng. PAS
Warsaw, Poland

1. Introduction

Human individuals exist in their home, social, professional and/or natural environments. A permanent exchange of information between us and the external world is a basic condition of supporting our vital activity. Unfortunately, there is also a big amount of people living among us that suffer from different sorts of disfunction of their perceptional /vision, hearing etc./ and/or effectorial /speech, moving, operating/ organs. Rehabilitation of invalids is a big human, social, economical and technological problem.

A progress reached in the last years in the domain of microelectronics and data processing technology opens new perspectives also for invalids rehabilitation. A general purpose is to help them in their contacts with external world in education, professional work and in home resting. The aim of this paper is to present some concepts concerning the above-mentioned problem. In particular, we shall show the way of using microcomputers as intelligent interface between an invalid and his environment that helps him communication and action in different situations.

2. General remarks about human receptory tracts

Vision, hearing, touch, smell and taste are the senses that are usually indicated as basic human tracts of information perception. It seems probable that some vestigial organs permet us to receive some other sorts of physical signals coming from external world. However, they play less important role in our life and will be not considered here.

The following factors are taken into account when talking about the importance of any sensor:

- mean even/or maximum information flow /bits/s/ transmitted to the central nervous system,

- a general vital role of the given sort of information received by the sensor.

From both points of view the order:

$$\text{VISION} \succ \text{HEARING} \succ \text{TOUCH} \succ \begin{cases} \text{SMELL} \\ \text{TASTE} \end{cases}$$

reflects the role we assign intuitively to the sensors /vision being considered the most important one/. However, a more exact evaluation of parameters characterizing, say, the information flow through a given sensory organ is a rather tremendous work. It is caused by the fact that a nervous connection between the sensory organ and the corresponding cerebral information processing centre is not simply a communication line but it usually plays a more active role in information processing. As a result, the information received by the central nervous system usually differs in volume and in form from this received by sensory organs located on the body's surface. In addition, we are able to control, by focussing our attention on any given sort of received signals, a relative information flow delivered by the sensory organs to our central nervous system and acting on our consciousness. Therefore, it is very difficult to say exactly how much of information there is received by the given sensory tract, so as it depends on basic methodological preassumptions. However, if considering the inputs of nervous connections between sensory organs and cerebrum we can say that vision of a health man delivers approximately 10^5 times more information per second than his hearing organs, hearing - 10^3 times more than touch and the last approximately 10^2 times more than smell or taste. Let us also remark that optical signals carry information from practically unlimited distances while touch and taste handle with information sources being close to our body; acoustic and smell signals are in the middle of those two extremities. For direct supporting vital processes information coming from our close environment seems more important. This fact explains why primitive biological organisms are very often dowered with taste and touch sensors only. On the other hand, our world cognition and understanding as well as social relations development we owe to our ability to receive information carried by optical and acoustic signals.

3. Informational aspects of invalidity rehabilitation

Let us focus our attention on some cybernetical and informational aspects of invalidity of sensory organs. Invalidity is here considered as a durable reduction or elimination of ability of a sensory organ to receive a given sort of signals and to transmit them to the corresponding cerebral centre. Therefore, the case of mental diseases or of central nervous system impairment making signals reception and understanding impossible is not considered here. The situation under consideration is thus illustrated in fig. 1.

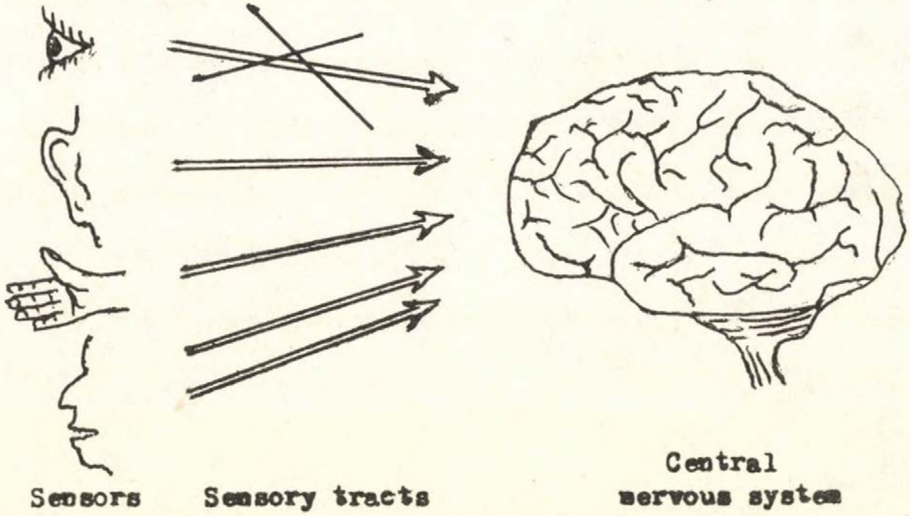


Fig. 1

There are several sensory tracts receiving different sorts of signals and transmitting them to the corresponding local centres where the information is recognized. The recognized patterns are then used for a global situation interpretation and understanding on a higher level of information processing in central nervous system. The problem arises what will be the results of a given sensory tract destroying. Generally speaking, if a given observed object manifests itself in different ways, its pattern /maybe simplified/ can be reconstructed on the basis of signals received by the remaining sensors. This fact becomes a big chance for the invalids that up to a certain degree can use some of their sound sensors instead of the injured ones. It is well known, for example, that we can recognize, in certain circumstances, a person not seeing him but hearing his approaching steps. Deaf-and-dumb children are teached to recognize spoken words by observation of lips movements. We can evaluate density and consistency of a liquid, without touching it, by hearing its splashes only, etc. However, substitution of injured sensory organs by the other ones is limited to the class of cases in which several kinds of signals carry the same information.

Transplantation of sensory organs into the place of the injured ones seems a promising alternative solution of the problem. Using glasses or hearing aids was a first step in this direction. The next one will be a development of surgery of eye and/or ear with transplantation of natural or artificial elements /like eye lens or eardrum/ and in the future - of complete organs including the sections of nervous tracts. Let us remark, however, that we are not able yet to construct artificial sensory organs /of any sort/ that are functionally equivalent to the natural ones. Therefore, transplantation of complete sensors can be considered only as a transplantation of natural organs.

At last, there is a third possible solution of the problem, being of interest for computer scientists: microprocessor-based aids for the invalids or artificial quasi-sensors adjusted to their sound sensors. A general idea of this solution is shown in fig. 2.

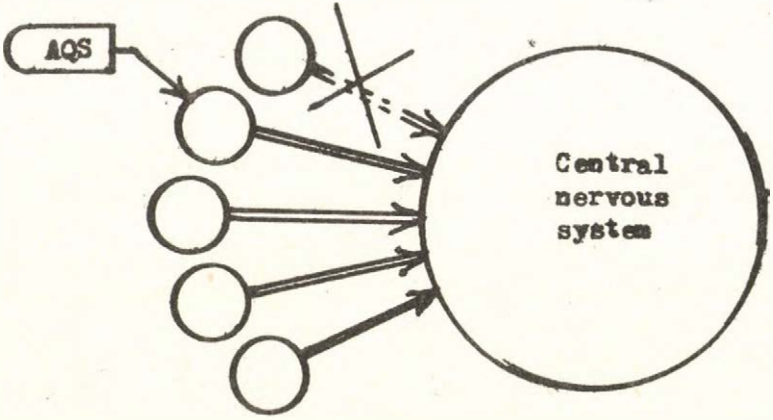


Fig. 2

In the simplest case an artificial quasi-sensor /AQS/ is a signal transformer that the signals destinated to the injured sensor transforms into the ones that can be received by a substitutive sensor. From a theoretical point of view this solution is possible, in general, if information flows don't overpass the information channel capacities. Therefore, using a sound vision tract for transmitting information that is usually transmitted by a hearing tract seems, at a first glance, a realistic concept, while the opposite seems rather impossible. However, the general law of non-overpassing channel capacities by information flows should be interpreted more precisely. The point is that information received by our sensors is highly redundant. Our visional tract architecture is based on a principle of convergency of inter-cellular connections and, as a result, of convergency of signals transported to the cerebral centre / [1] , 11.3/. A similar principle can be observed also in our aural organs / [1] , 12.2/. In addition, it has been observed that the sort of signals that activate the neurons in our sensory tracts changes along the tracts: it corresponds to elementary signals in the close environment of sensors and to some higher-order compositions of signals when approaching to the cerebral centre. This is the key to the construction of computer aids for invalids: we can use sensory tracts with limited information capacity for transmitting signals of any primary form if the last is selected and reduced in a corresponding manner. Therefore, construction of microprocessor-based AQSs is not only the problem of signals conversion but rather a one of signalsselection and transformation into a suitable form. In addition, by psychological reasons very specific ergonomic demands as well as technological constraints are imposed on the AQSs. They must be small, handy, reliable, easily energy supplied etc. This.explains why microprocessors seem very perfect elements for such devices. The AQSs can be divided into two general classes:

a/ portable AQSs using limited numbers of simple information processing algorithms,

b/ stationary AQSs equipped with a library of more sophisticated programs that may help the invalid in education, work or resting.

In both cases the following scientific and technological problems arise:

i. construction of suitable, versatile and reliable signal receptors and effectors;

ii. invention of real-time signal processing algorithms;

iii. elaboration of simple and effective control and manipulation methods, acceptable by the users of AQSs;

iv. extension of AQSs' abilities by equipping them with adaptive and/or self-organizing mechanisms.

It should be remarked that the best AQS is the one that is felt by the user as a natural extension of his natural abilities and doesn't need too much attention for being handled. That is why the most sophisticated solutions very often are not the optimum ones.

4. Remarks about artificial receptors and effectors

Up-to-date signal processing technology gives us a large variety of physical signals reception devices, extending our natural abilities in the domain of reception of infra-red or ultraviolet optical signals as well as infra- and ultraacoustic signals. We can also detect and evaluate lot of other physical parameters directly not affecting our sensors, like connected with steady magnetic fields, ionization levels, corpuscular radiation etc. However, we cann't meet the competition with Nature in making artificial receptors as versatile and miniature as the natural ones. In the following table there are given some examples of artificial receptors.

Kind of signals	Artificial recepters
Optical	Photocells, photocell rows or matrices Photoelectric multipliers TV cameras: - monochromatic - polychromatic /RGB/
Acoustic	Single-frequency acoustic sensors Wide-band rows of acoustic sensors Acoustic time-functions analyzers
Mechanical	Electromechanical, electroresistive or piezoelectric sensors of - deflection - pressure Rows or matrices of mechanical sensors
Chemical	Ion-sensitive field-effect transistors

Some of the above-mentioned artificial receptors cann't be recommended but for stationary AQSs only /say, the polychromatic TV cameras/. Some other ones are of less importance, like the ion-sensitive FETs which can be used in artificial taste sensors.

In a similar way there can be specified the effectors or output devices of AQSs, as shown in the following table.

Kind of signals	Artificial effectors
Optical	Lamps Visual indicators: - lamp matrices - digital displays - liquid crystal displays Electromechanical indicators Screen monitors: - monochromatic - polychromatic /RGB/
Acoustic	Buzzers, ringers Headphones, loudspeakers Acoustic synthesizers
Mechanical	Electromechanical stick or membrane deplacement indicators Electromechanical or piezoelectric vibrators Electromechanical or piezoelectric stick or vibrator rows or matrices

Mechanical effectors can be located on hands, thighs, forehead, shoulders etc. In portable AQSs small and hidden effecters are usually preferred.

5. Signal processing algorithms

Choosing signal processing algorithms for AQSs should be based on a previous analysis of situations in which the given AQS will be used. It is not sufficient to answer the questions: what sort of signals should be received and what is the suggested sort of substitutive sensor that will be used instead of the injured one. The next question is: in what conditions the signals will be received and what kind of interference will disturb the reception. It is evident that AQSs, in general, cann't substitute natural sensors but in singular and rather specific situations. The next problem arising is thus selection of signal processing tasks that are preferred as being aided by the AQS. Any task of this sort should be, in addition, characterized by the desired accuracy of solution as well as by its admissible response-time /so as being solved by the AQS in real-time/. For example, for blind people the following tasks can be specified:

Task characteristic	Accuracy	Response-time
Detection of large moving objects	middle	< 0,3 s
Evaluation of distance to large stationary objects	low	< 2,0 s
Detection of objects of given shape and size	middle	< 1,0 s
Recognition of indications on displays	high	< 1,5 s
Reading printed texts:		
- for a fixed alphabet	high	~ 20 char/s
- for several collections of characters	middle	~ 15 char/s

Here "low" accuracy means more than 30% of erroneous answers, "middle" accuracy - between 10 and 30% and "high" accuracy - less than 10%, however, the numbers has been taken in an arbitrary way.

It is also evident that the last two tasks can be recommended for stationary visual AQSs rather while the former ones can be realized by handy, portable AQSs.

Most of the above-mentioned tasks belong to the class of pattern recognition problems and can be solved using the well-known methods based on deterministic, statistical or structural approach / [2] , [3] /. The real-time signal processing condition leads to some additional technological problems, so as the received optical signals should be stored in an image buffer-memory before processing. The specific technological problems will be not considered here.

In similar way, for deaf and/or deaf-mute people the following signal-processing tasks can be offered:

Task characteristic	Accuracy	Response-time
Detection of any loud burst of noise with indication of direction	middle	< 0,2 s
Recognition of typical acoustic signals /doorbell, human voice, dog's bark etc./	middle	< 0,5 s
Recognition of selected single words spoken	middle	< 0,5 s
Loud reading of short texts introduced through a keyboard	high	< 0,1 s
Displaying in typed form spoken texts	middle	< 1,0 s

Effective solution of some of the above-mentioned tasks needs using advanced information processing methods. For example, displaying continuous spoken texts is a problem that concerns speech analysis on morphological, syntactical and sometimes on semantical levels. The problem needs thus high data processing rate as well as high RAM's capacity.

All the above-mentioned signal-processing tasks were based on an assumption of steady working conditions in which using fixed algorithms and programs is possible. However, the AQSs based on such principle would be rather unflexible and in fact uneffective. The working conditions may change due to the changes in the environment the invalid exists and acts as well as due to the changes of his proper needs. In particular, it is necessary to take into account the fact that the abilities of invalids to solve their vital problems develop according to the experience stored, to the health state etc. This means that the AQSs should be flexibly programmed and able to be adapted to changing circumstances.

So as the AQSs perform their signal-processing operations in real time, they are programmed so as to repeat their programs in loops with repetition-time not overpassing the admissible response-time of the AQS. However, in order to make the programs adjustable to the current user's demands it is defined up to a certain set of values of parameters controlled by the user. The parameters can be connected with

i. calculation constants, like: signal threshold levels, time-intervals duration, frequencies, signal-shape characteristics etc.;

ii. program constants, like: dimensions of arrays, number of repetitions in program loops, subroutine labels etc.;

iii. structural algorithm constants, like labels assigned to optional connections within a general logical scheme of a program etc.

The set of admissible values of adjusted parameters determines a sort of a space of control signals for the user of AQS. The parameter values can be set by keys and stored in registers for being used during the program performance. From the user's point of view the keyboard states thus form a sort of expressions of a manipulation language that can be used for AQS hand-control. However, it is assumed that the user is not able to set his own programs of signal processing, the last being designed by the producer of AQS. It is also clear that the degree of freedom in program adjusting offered to the users is larger in stationary AQSs than in the portable ones.

Future progress in AQSs construction will concern their ergonomic parameters, reliability, versatility, sensibility and accuracy. The manipulation language will be "naturalized" so as to make AQS control easier and more flexible. This will be reached due to wide application in AQSs construction all achievements in microelectronic schemes technology, in pattern recognition and artificial intelligence and in construction of signal receptors.

References

1. R.F. Schmidt, G. Thews. Human Physiology. Springer Verlag, Berlin, 1983.

2. J. Kulikowski. Cybernetyczne układy rozpoznające. PWN, Warszawa, 1972.

3. K.S. Fu. Syntactic methods in pattern recognition. Acad. Press, New York, 1974.

ON THE PADRE'S IMPLEMENTATION BY PREPROCESSOR TECHNIQUE

Van-Lu NGUYEN

LITP et Université Pierre et Marie Curie
4 Place Jussieu
75252 Paris Cedex 05

Thuan HO

Computer and Automation Institute
Hungarian Academy of Sciences
Victor Hugo St. 18-22, Budapest

ABSTRACT

Different steps and advantages of preprocessor technique are showed in this paper by way of a concrete implementation of Padre, a programming language especially designed for describing, transforming and interpreting various classes of Petri nets. We also show that the technique is perfectly applicable on microcomputers.

1. INTRODUCTION

It has been exposed in [1] that Padre is an experimental programming language which can describe, transform and interpret different classes of Petri nets - e.g ordinary nets, capacity nets ([2]), coloured nets ([3]), predicate/transition nets ([4]), ... In this paper, a particular Padre's implementation technique is exhibited.

In this implementation we seek two objectives: development speed and portability. The first one can be obtained by preprocessor technique which translates a high-level source language into another high-level target language. The semantics of the source and target languages are often well-understood, so the translation implies no major difficulties because the existence of their direct semantic equivalence. For portability objective, we have to choose a language which is universally implemented on almost all computer families. And that is the case of the language Pascal.

With these objectives in mind, our preprocessor technique consists in four steps:(i) internal representation generating of different types with pertinent informations for future uses, (ii) translating all Padre program into a semantically equivalent Pascal program, (iii) generating useful routines in prevision of user's needs, and (iv) submitting the outputs of preceding steps to a Pascal compiler which completes the translation. The preprocessor design includes of course all major and well-known algorithms of compiling processus ([5]) - i.e lexical and syntactical analysis, semantic analysis, error recovery, ... - except perhaps that the generated object is coded in high-level language.

The paper is organized in three parts: in the first one, we recall some basic constructs of the language Padre, the second part describes our implementation technique, and in the third part we show that this technique is perfectly applicable on micro-computers.

2. PADRE LANGUAGE SUMMARY

In the following language summary, we only recall some salient constructs of Padre. One can find a compact description of Padre in [1] or its description in-extenso in [6].

To remedy the incommodity of current high-level languages in the description, transformations and interpretations of various Petri net classes, we have designed Padre equiped with the following type system and set of instructions and primitives:

2.1 type system

Beyond elementary and standard types such as char, boolean, integer, and integer subrange, Padre offers a net constructor with the following syntax (from this time forward the EBNF ([7]) is used):

```
net_id = "net of"
             [const_def ";"]
             [type_def  ";"]
             element_decl";"
             connection_clause
         "end"

element_decl = "element" elem_item {";" elem_item}
elem_item = elem_id {"," elem_id} ":" elem_struct
elem_struct = "transition" | "simple place" |
              "colored place of" int_subrange_id
connection_clause = "connection"
                             trans_connect [interface]
                         {"|"trans_connect [interface]}
```

The trans_connect has the following simple syntax

```
trans_connect = trans_id "["
                          [in_clause";"]
                          [guard_clause";"]
                          [out_clause]
                       "]"
```

in which and for each transition identifier trans_id:

- in_clause specifies all its in_places and its associated preconditions,

- guard_clause indicates either an additional constraint on variables forming its preconditions or its synchronization interface with the outside world,

- out_clause specifies all its out_places along with the associated postconditions,

- interface specifies the interface with the outside world.

Instead of going further in syntactic details which are been completely described in [6], let us give a net description example already found in [1]:

Example 1.

The net in fig.1 represents a solution without inhibitor arc of the classical readers and writers problem ([8]). The access coherence implies three well-known constraints: (i) writings are mutually exclusive, (ii) concurrent readings are allowed, and (iii) readings and writings are mutually exclusive.

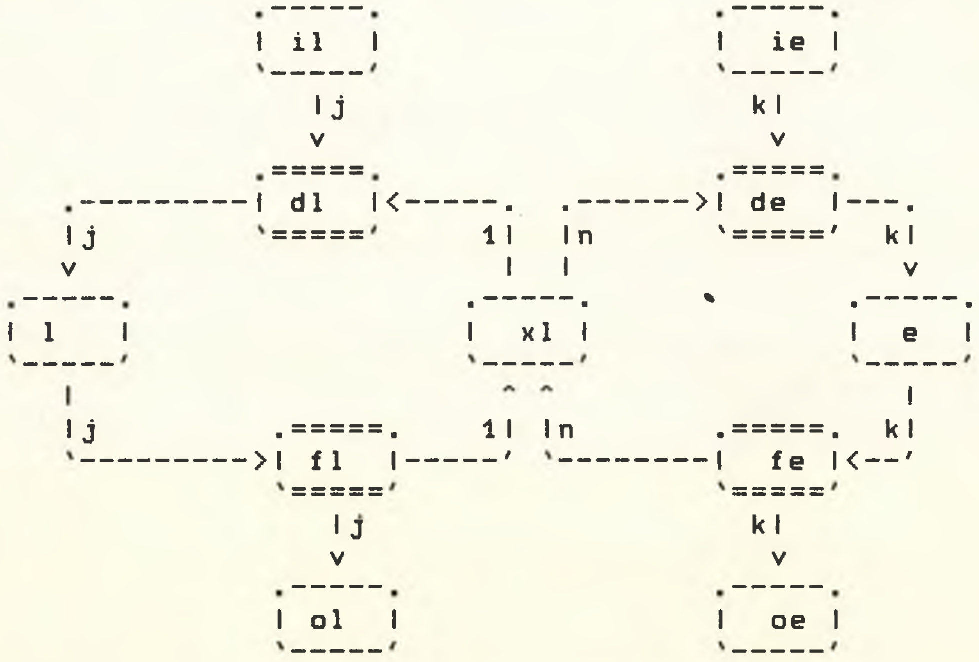


fig.1: rwcolor

where

il (respectively ie): place containing all processes which are waiting for a reading (respectively writing),

l (respectively e): place containing all processes which are reading (respectively writing),

ol (respectively oe): place containing all processes which have terminated their reading (respectively writing),

xl: simple place,

dl (respectively fl): transition which represents a reading start (respectively end),

de (respectively fe): transition which represents a writing start (respectively end).

We are evidently in the presence of a coloured net which can be described in Padre as follows by the preceding net constructor:

```
const n=13;
type np=1..n;
 rwnet=net of
        element
         il,l,ol,ie,e,oe:colored place of np;
         xl:simple place; dl,fl,de,fe:transition;
         connection
          dl [ in xl:(xl>=1), il(j:np):(il[j]>=1);
               out l:(1 to l[j]) ] |
          fl [ in l(j:np):(l[j]>=1);
               out xl:(1 to xl), ol:(1 to ol[j]) ] |
          de [ in xl:(xl=n), ie(k:np):(ie[k]>=1);
               out e:(1 to e[k]) ]  |
          fe [ in e(k:np):(e[k]>=1);
               out xl:(n to xl), oe:(1 to oe[k]) ]
       end;
```

From the basic net constructor, following operations are allowed on nets:

- *operations intra-nets*: these operations allow adding new elements to a net, deleting elements of a net.

- *operations inter-nets*: that are operations such as combination of two nets by merging their identically named elements, combination of nets by merging their explicitly named elements, combination of nets by merging two place sets.

2.2 set of statements and primitives

This set includes the following statements and primitives:

classical statements and primitives

- assignment statement which is classically defined as

```
        variable ":=" expression
```

- conditional choice specified by an if_statement:

```
        "if"
            guarded_command {"|" guarded_command}
        "fi"

        guarded_command = guard ":" statement_sequence
        guard = boolean_expression
        statement_sequence = statement {";" statement}
```

In an if_statement, each guarded_command is examined: if its guard is true then the associated statement_sequence is executed.

- non-deterministic loop specified by an loop_statement:

```
        "loop"
              guarded_command {"|" guarded_command}
        "pool"
```

In a loop_statement, each guarded_command is *randomly* examined: if its guard is true then the associated statement_sequence is executed. The loop_statement terminates when all guards become false. The if_statement and loop_statement are inspired from Dijkstra's guarded_commands ([9]) with a slightly modified semantics.

- input/output primitives: all Pascal's input/output primitives are included in Padre with the same semantics.

proper primitives for nets

- creating an instance of net: the primitive

```
        "create" "("net_var_id ":" net_type_id ")"
```

creates a net instance, identified by net_var_id, with the type specified by net_type_id.

- destroying a net instance: the primitive

```
        "delete" "(" net_var_id ")"
```

destroys the net instance identified by net_var_id and restitutes the occupied memory to the available memory space.

- random interpreting of net: the primitive

```
        "randomexec" "(" net_var_id ")"
```

activates a random interpreting of the net instance net_var_id. The interpretation terminates when all transitions have been proposed as candidates for firing. Because of random choices, one transition can be proposed many times.

- interpreting of net with priority: the primitive

  "priorityexec" "(" net_var_id ")"

activates a net interpretation in which transitions are selected as candidates according to their relative priority.

The following two primitives allow interactive handling and interpreting of nets:

- the primitive

  "stepbystep" "(" net_var_id ")"

gives access to the net instance net_var_id and offers the following interactive facilities:

  - net marking display, net connection matrix display,
  - net place list and/or net transition list display,
  - display of instantanously fireable transitions,
  - accessing a place or transition in order to modify and/or test its attributes.

- the primitive

  "netswork" "(" net_var_id1 "," net_var_id2 ")"

offers the following operations on two net instances specified in parameters:

  - merging of two places or two transitions,
  - merging of identically named elements,
  - merging of two place sets ([10]).

3. PADRE'S IMPLEMENTATION

This implementation can be described by the following scheme:

```
.---------------------------.
|       Padre program       |
`---------------------------'
             |
             |
             v
.---------------------------.
|    Padre's preprocessor   |-----------------.
`---------------------------'                 |
          |              |                    |
          |              |                    |
          v              v                    v
.-----------------.    .-------------.    .--------------.
| representation  |    | equivalent  |    |    Padre     |
|  of net types   |    |   Pascal    |    |   library    |
`-----------------'    |   program   |    `--------------'
        |              `-------------'           |
        |                    |                   |
        v                    v                   v
.--------------------------------------------------------.
|                     Pascal compiler                    |
`--------------------------------------------------------'
                          |
                          v
                  .--------------.
                  | object code  |
                  `--------------'
```

We can subdivide the implementation in two steps: in the first one is described the translation of the Padre's declarative part and in the second, the translation of its imperative part.

translation of declarative part

The translation of primitive standard types - i.e char, boolean, integer, integer subrange - do«not imply any major difficulties thanks to the syntactical analogy between Pascal and Padre.

The translation of net type constructor necessitates, on the other hand, a deep analysis of net type descriptions to dynamically generate internal representations of nets. To anticipate the using of large nets, all generated representations are stocked in secondary memory.

For each net type description, the translator generates then a corresponding internal representation, a more or less complicated graph, in which all pertinent informations for future uses - such as preconditions, guards, postconditions, interfaces with outside world - are appropriately transformed and explicitly retained.

For each intra- or inter-net operation, there is in the Padre's library a corresponding algorithm purposely elaborated. So the library inludes at least a collection of graph-manipulating algorithms such as:

- graph-traversing algorithm which is useful, for instance, to generate pre- and post-conditions of each transition,

- graph-duplicating algorithm, useful to generate net instances,

- graph-merging algorithm to realize net combinations,

- garbage collecting algorithm for memory recovery when a net instance is destroyed,

and so on.

translation of imperative part

Generally the translation of this part do not implies any major difficulties because there is a direct semantic equivalence between Padre's and Pascal's control structures. Except, perhaps, for guarded commands and for net handling and interpreting primitives.

Let us describe the translation of these cases:

translation of guarded commands

- if_statement

An if_statement of Padre

```
if
  guard_1 : stat_seq_1 |
  guard_2 : stat_seq_2 |
  .....
  guard_k : stat_seq_k
fi
```

can be translated into the following Pascal sequence:

```
if T(guard_1) then begin T(stat_seq_1) end;
if T(guard_2) then begin T(stat_seq_2) end;
......
if T(guard_k) then begin T(stat_seq_k) end;
```

where T(<anything>) represents the translation result of

<anything>.

- loop_statement

A loop_statement of Padre

```
loop
   guard_1 : stat_seq_1 |
   guard_2 : stat_seq_2 |
   ...
   guard_k : stat_seq_k |
pool
```

is translated in the Pascal sequence:

```
j:=0;
repeat
  case j of
       0 : begin end;
       1 : if T(guard_1) then begin T(stat_seq_1) end;
       2 : if T(guard_2) then begin T(stat_seq_2) end;
       ...
       k : if T(guard_k) then begin T(stat_seq_k) end;
  end {case};
j:=alea(generator,k);
until not ( T(guard_1) or T(guard_2) or
              ...     or T(guard_k))
```

In this translation scheme, the non-determinism of loop_statement is reflected in a random choice of guarded command to be executed. In a syntactical analysis with one symbol look-ahead technique, the number of guarded commands is unknown at the loop_statement entry; in order to apply this technique in our translation, we have added a dummy statement

```
0 : begin end;
```

translation of net handling and interpreting primitives

The translation of net instanciating and destroying primitives is easy because each one corresponds to a library's algorithm, for example:

- a net instanciating activates a graph-duplicating algorithm,

- a net deleting activates a garbage collecting algorithm.

The translation of various net interpreting mechanisms necessitates, on the other hand, two basic routines, the first one for firing of a transition t which can be described by:

```
                 firing_a_transition_t

i): { if t do not have any coloured in-place then the
  following Pascal sequence is generated:}

"if" [guard_of_t "and"] precondition_of_t "then"
"begin"
    [interface_activation ";"]
    mark_consuming_sequence";"
    mark_adding_sequence";"
"end"

ii): { if t has at least one coloured in-place then the
      following sequence is generated in which parameters
      x, ..., w represent colours of the t's in-places.
      The interpreting algorithm does then an exhaustive
      search to test the truth value of t's precondition
      and guard}

 xt:=alea(generator,max_of_type_of_x);
 x:=xt;
 ...
 wt:=alea(generator,max_of_type_of_w);
 w:=wt;
 repeat
    x:=x mod max_of_type_of_x + 1;
    ...
      repeat
         w:=w mod max_of_type_of_w + 1;
         if [guard_of_t and] precondition_of_t then
         begin
            done:=true;
            [interface_activation;]
            mark_consuming_sequence;
            mark_adding_sequence
         end;
      until (w=wt) or done;
      ...
  until (x=xt) or done;
```

The second routine interprets a list of transitions pointed by p:

```
{ interpret the list p}

while p<>nil do
begin
   t:=transition_pointed_by_p;
   firing_a_transition_t;
   p:=transition_following_p;
end;
```

With these two routines, the other primitives are easily translated, for example:

- the primitive *randomexec* is translated into

```
R:=[];
repeat
   t:=choice(T);
   firing_of_t;
   R:=R+[t];
until R=T;
```

where T is the set of net transitions, R the current set of randomly choosen transitions, and choice a random choice algorithm.

- the primitive *priorityexec* is translated into

```
generating a list p of transitions according to their priority;
interpret the list p;
```

The translation of *stepbystep* and *netswork* implies all apparatus previously evocated.

4. CONCLUSION

The two previously stated objectives in this Padre's implementation by preprocessor technique have been attained:

- development speed: the preprocessor is completely written and now operating on MULTICS system. Its implementation has necessitated 12 months/man for about 10000 Pascal lines.

- portability: the transfert of the system on VAX/UNIX is now undertaking, it doesnot seem imply any problem and will be operating very soon.

The current program text of the preprocessor can be decomposed as follows:

- 172K bytes for the source code written in Pascal,

- 138K bytes for the Padre's library written in Pascal,

- 232K bytes the object code after link edit.

So this implementation technique is well-suited for almost micro-computers currently in uses.

REFERENCES

[1] Van-Lu NGUYEN
"Padre: A Petri nets based parallel programming language"
in Proceedings of The 4th Hungarian Comp. Sci. Conference
Gyor, July 1985 Budapest
Eds. M. Arato, I. Katai, L.Varga

[2] Charles ANDRE
"Systèmes à évolutions parallèles: Modélisation
par réseaux à capacités et Analyse par abstraction"
Thèse d'Etat, Université de Nice Février 1981

[3] K. Jensen
"Coloured Petri nets and the invariant-method"
in Theoretical Computer Science no 14, 1981

[4] H. J. Genrich and K. Lautenbach
" The analysis of distributed systems by means of
predicate/transitions nets"
in Semantics of concurrent computation
Lecture notes in Somputer Science no 70

[5] N. Wirth
"Data Structures+Algorithms=Programs"
Prentice_Hall 1976

[6] Van-Lu NGUYEN
"Padre: un langage d'expérimentation de programmation
parallèle et distribuée basé sur les réseaux de Petri"
Thèse d'Etat, Université Paris VI à paraitre 1986

[7] N. Wirth
"What can we do about the unecessary diversity of notation
for syntactic definitions"
in Comm. of ACM November 1977

[8] P.J. Courtois et al.
"Concurrent control with readers and writers"
in Comm. of ACM October 1971

[9] E. W. Dijkstra
"Guarded commands, nondeterminacy and formal derivation of
programs"
in Comm. of ACM August 1975

[10] V. E. Kotov
" An algebra for parallelism based on Petri nets"
in Lecture notes in Computer Sciences no 64, 1978

PERSONAL COMPUTER-AIDED TESTING AND TRAINING SYSTEMS

Radoslav Pavlov, Ruslan Mitkov, Avram Eskenasi

Insitute of mathematics, Sofia

ABSTRACT

In the present paper are examined the basic principles of achievement testing, the historical development of its computerization and mostly some computer-aided testing and training systems developed at the Institute of mathematics of the Bulgarian Academy of Sciences. The systems were developed on the personal computer "Apple-II". Certain advantages of the computer-aided testing and training (the latter realized efficiently only by means of computer), as compared to the traditional testing and training are examined, as well as future problems and tendencies of computer-aided testing and training.

ACHIEVEMENT TESTING

An achievement test is a systematic procedure for determening the ammont a student has learned. Testing has been around for centuries, for longer than most people today realize (ancient China). An widespread and rapid development of testing procedures and generally of theory of testing was noticed at the end of the 19th and the beginning of 20th century and the names of Rice, Thorndike and Binet deserve to be mentioned.

Achievment tests are widely used in education as a suitable instrument to measure someone's knowledge. The tests can be classified as essay, problem and objective tests. The objective tests can be scored more rapidly and reliably than either of the other types (especially preferable when large amounts of students are to be tested). Usually a test is a collection of test items. Objective items may be : true-false, multiple-choice, matching and short-answer.
It is certain, that multiple-choice test items are currently the most

highly regarded and widely used form of objective test items. They are adaptable to the measurement of most important educational outcomes - knowledge, understanding, and judgement ; ability to solve problems, to recommend appropriate action, to make predictions. It was the multiple-choice objective test items , that drew the attention of the informaticians in the development of the first computer-aided testing programs and systems. Where ever testing is concerned, we may speak also in terms of "training" (drill and practice), provided there is appropriate feedback.

LOOKING BACK : THE FIRST STEPS IN COMPUTERIZATION OF TESTING

The first computer programs(and later systems) were CTSS (USA, 1969) , MEDSIRCH (CANADA, 1970) , MENTREX (USA, 1973). These programs permitted item entry and storage (in the socalled "Itembank"), item bank maintenance, item generation and item scoring. In the following years in several countries (USSR, Japan, Bulgaria) were developed systems with more flexible and extensive possibilities. The systems TEST-1, TEST-2 and TEST-3 were developed in 1977 at the Institute of mathematics of the Bulgarian Academy of Sciences. At the beginning, the test items were kept on punch-cards and magnetic tapes. Later on, with the development of computer technology, it was possible to ensure direct access to the stored on hard discs test items. The appearance of personal computers with their new interactive(dialogue) and graphics possibilities gave a strong impact to the development of computer-aided education . In the field of computer-aided testing and training it was possible to think of improvements of all the stages connected with testing - test development, test administration, test scoring and reporting practices, item analysis and with training - the emergence of new training individual strategies, including some adaptive features.

These ideas were incorporated in the from us developed microcomputer testing and training systems.

UTEST AND UTRAIN - COMPUTER-AIDED TESTING AND TRAINING SYSTEMS

The systems can be described in two different ways : from viewpoint of the user-to be examined or trained (here called "student") and from the viewpoint of the user-examinator or trainer (here called "teacher"). From viewpoint of the student UTEST may be described as follows : the student sits in front of the microcomputer, receives brief instructions (several sentences on the screen) and is offerred a test, consisting of multiple-choice test items. The test item has a problem situation and several (3,4 or 5) alternatives, one of which correct and the others - distractors. The test item may also contain graphical illustration to the problem situation. The student is supposed to select the correct answer (choosing for instance "1", "2" or "3") or delay his answer. If after the generation of the test and its display on the screen the time is not up, the delayed items are offerred sequentially again. When no more items available, the results are stored in a file, further accessible to the teacher.

UTRAIN is a modification of UTEST on the basis of drill and practice mode. The student may have immediate feedback after answering each item ; the time, determined for examination may be adjusted according to the individual pace of work ; there is no assigning notes or mastery or nonmastery to each student, but just drill and practice covering desired subject areas.

From viewpoint of the teacher UTEST may be regarded functionally: UTEST consists of four larger subsystems : text base maintenance (text base editor), graphics base maintenance (graphics editor), test generation and examination and results processing. These systems are functionally connected through different files.

UTEST uses floppy discs and this brings certain restrictions of the data base size and technological complications of the results processing (additional complication of the latter is that the computers are not connected in net).

Special database maintenance means are elaborated for the examinator. Text and graphics are stored separately and supported in different ways. The text base maintenance means (text editor) represents an elementary menu and no preliminary knowledge is necessary to correct, update or eliminate text. The graphics base maintenance means (graphics editor) enables easy creation of various colour figures, which occupy minimal memory space and their maximally fast execution. To make use of the graphics editor is not hard to learn. The teacher is in position to require the generation of a test from the data base with determined characteristic features - number of test items, subject areas, global difficulty, time for test solution. Each test item is characterized by "difficulty weight" and belongs to certain subject area.

It should be mentioned, that UTEST is one of the not so many really functioning testing systems. We have managed to realize UTEST in several different subject areas. The most popular concrete version of UTEST is KATEST, a testing system for administration of the motor vehicle driver's exam (4) . KATEST is a menu-driven testing system. Its main menu looks like that :

1. Examination
2. Results processing
3. Text base maintenance
4. Graphics base maintenance
5. Copying of diskettes
6. Exit

Option 5. is necessary because of the special protection measures taken (KATEST works under completely different, specially elaborated DOS). They were a number of experiments, carried out with KATEST. We were glad to establish , that KATEST was friendly accepted by most of the examinees.

A more detailed functional scheme of KATEST is given in the appendix (page 13).

KATEST was not the only concrete realization of UTEST. For the students of Sofia University we developed testing systems for biology, physics, mathematics and law. We are working now on some perfection of UTEST, which is going to bue used by the Sofia Council of People's Education. We plan to include some powerful information subsystem , giving any information concerning the "achievement history" of each student. Furthermore, item analysis should be carried out by the system.

The authors are working now on a project, which will connect a group of computers in net and use common big capacity disc. The present paper will not discuss the advantages of such solution.

Comparing UTEST with traditional testing, we may state the following advantages in favour of UTEST :

- The number of tests offerred on traditional answer-sheets is not very large. This enables sometimes students learning tests by heart in advance and so arises the problem of test security ;in UTEST the test items are selected within each subject area by random generator - it means, that provided the data base is large enough, practically an unlimited number of distinct tests are produced, not known in advance by students.
- It is very difficult and expensive to change the tests in

traditional way. Only a single change of an item, resulting from certain pedagogical or administrative reasons leads to the print of numerous tests. In UTEST all is simply done : the respective item is retrieved and by means of the text editor corrected.

- a similar problem arises when the structure of the test should be changed.In UTEST the structure of each test may be predetermined by the teacher.

- the advantage of UTEST is similarly seen when the global weight of the test should be changed

- the subjective factor is in UTEST fully eliminated.

It should be also mentioned, that UTEST is independent on the subject area, i.e. universal with regard to it. The user-friendly menu enable teachers even with little or no knowledge in programming to create their own data base.

In UTRAIN there exists the same functional structure. However UTRAIN has more options connected with the individual preparation of each student. The student is allowed to select a desirable teststructure for the drill and practice, to select the time for it, to select the mode of feedback. These options can be also selected by the teacher. Furthermore UTRAIN may analyze the level of knowledge of each student, instead of simply assigning a corresponding note to it.

ADAPTIVE FEATURES

The authors are trying to incorporate some "adaptive features" in the systems. Adaptivity is realized in examination differently from drill and practice. Essential in both cases is the subject determination of the test items. After some analyses and hesitations the authors preferred the hierarchical classification scheme (HKS) as more suitable than describing each test items by means of key words. The HKS contains 6 levels, and according to the properties of this type of languages each subject area is described by code (from 1 to 6 digits) and each subject area coded by k digits represents a subarea of each area, coded by the leftmost m digits of its code ($m < k$). Each test item belongs exactly to one subject area. Here is a simple example of it :

1. Mathematics

1.1 Elementary mathematics

1.1.1. Arithmetics

....

1.2 Higher mathematics

1.2.1.........

1.2.10 Probability theory

1.2.10.1. Random events

1.2.10.2 Random variables

....

2. Informatics (computer science)

2.1. Programming

2.1.1 Programming languages

....

3. Mechanics

The authors are acquainted with some attempts aimimng to set up adaptive CAI systems. Some of these systems are based on creating a model of the student and adapting the CAI (computer-aided instruction) in connection with this model . Others make use of sophisticated statistical theories for estimating student's ability. In this paper the authors describe their aim to obtain adaptivity in two strictly defined and not so complicated sides of the education - examination (testing) as well as drill and practice (training). What is more, they aim at obtaining this adaptivity by maximally elementary and user-accessible means.

The simplest adaptivity means in UTEST represents the possibility to postpone the answer of any test item. **We can talk** in this case about adaptivity and psychological adjustment of the student. The experiments, carried out, show that this possibility is frequently made use of. Unfortunately the authors do not claim any final conclusions, for too large statistical information is not available. This remark is also valid for the rest of the examined problems here.

Another adaptivity strategy is tealized in UTRAIN based on HKS. The teacher determines a structure of the drill and practice (just like in case of examination) : subject areas A_i, n_i , such that $n_i \leq card(A_i)$, total difficulty weight of the items of each area - M_i. Besides, a treshold number T_i for each area ($T_i \leq M_i$) and a number of loops N_i for each area are added. In case the student reaches at least T_i of given area, he receives items from the next area A_{i+1} . If the treshold number is not reached, the test items are presented once again on the screen with indication of the correct answers. After reading the correct answers, another n_i Items of A_i are generated. This process is repeated until the student receives at least T_i points, but not more than N_i times. In case of N_i unsuccessful attempts the exercise is suspended and a standard text is given, which recommends learning of the respective subject matter. If the subject areas are not closely related the drill and practice may carry on after registration of poor perfomance in the respective area. Through this simple mechanism (both realization and teacher's use) the system adapts itself to the individual peculiarities of the student.

Another attempt of adaptivity is to react to certain boredom of the students. This is especially important in case of children.

It is applied when a large number n_i of some area A_i (for example above 20) are available. Test items of equal difficulty are generated (In our systems difficulty range from 1 to 9, but practically only 1,2 and 3 are used). Suppose the result of the first third items corresponds to the result between T_i and M_i (i.d. result S_{i1} : $T_i/3 \leq S_{i1} \leq M_i/3$ is reached) and the results of the next sixth of elements falls down considerably (i.d. $S_{i1} - 2S_{i2} = = \dot{S}_{12}$, where S_{12} is a positive number). Then we could presume, that result deterioration comes from certain boredom and fatigue. In this case the drill and practice can be temporarily suspended and some music or animation can be offerred by the computer. The questions concerning the proper value of S_{12} and its parameters (regarding S_{12} as a function) are still open. A large number of experiments are necessary in order to establish if the selected values 1/3 and 1/6 are relevant and in general how efficient this approach is.

If boredom and fatigue, there might be another way of reaction to the high intermediate results. After reaching high number of points in the first third of items, the system generates the second third of items with higher level of difficulty. In this case the treshold number is automatically raised. The new treshold number serves as a criterion if the difficulty level of the last third of items should be further raised.

Another alternative of the explained procedure of UTRAIN may be applied in training students to obtain automatic and quick reactions in some specific areas. In this case instead of raising the difficulty level , time limit for answering each test item and in respect to the whole exircise is introduced.

So far as the examination is concerned, the authors are trying to realize in UTEST the following idea (which requires however more complicated structure of the data base in respect to the subject areas) : subject areas are not regarded as hierarchy, but as semantic net knots. This is a convenient way to express more adequately two, close in respect to the subject, areas (subject proximity). The authors aim by means of this net and the notion of subject proximity at establishing whether certain poor result is casual. It is proceeded in the following way : let S_i ($S_i < T_i$) be a poor result in A_i. Then S_i is registered, but several items from subject proximity areas of A_i are generated (according to the net). Poor perfomance in these close areas means unsatisfactory preparation in general. The disadvantage of this approach consists in overcharging the student. A palliative in carrying out this additional examination might be in doing it after the basic examination.

The authors regard their attempts to create an adaptive systems as in initial stage, at least because sufficient statistical information is not available and because no cooperation with psychologists and pedagogues is yet established. They hope to overcome these difficulties.

FUTURE PROBLEMS

Perhaps the basic problem of the future in computer-aided testing and training turns out to be the so far unsolved problem of computer-aided test item construction. This problem has attracted some methods for its partial solution (9) , (10) , but has not been completely solved yet. For its complete solution are needed most powerful means of the artificial intelligence.

An unambiguous answer should be given to the question to what extent the newest possibilities offerred by the microcomputers - motion, sound etc. can be used in the test items.

The restricions, established by computers and informatics exist practically no more. There exist almost no restrictions concerning the display of formulae, built of symbols of practically arbitrary alphabet, of sophisticated colour illustrations, accompanied even by motion and sound. For the first time it proves out, that not computer technology, but psychology and pedagogy are lingering behind.

Also, particular attention should be paid to the development of new and efficient adaptive strategies.

All these problems may be successfully solved, only if appropriate scientific cooperation between computer scientists, psychologists, pedagogues, linguists is established.

REFERENCES

(1) Ebel R. - Essentials of educational measurement, Prentice-Hall Inc., 1972

(2) Popham J. - Modern educational measurement, Prentice-Hall Inc.,1981

(3) Pavlov R., Eskenasi, Mitkov R. - An adaptive system of children examination and training, Proceedings of the international conference and exhibition on children in an information age : tomorrow's problems today, Varna, 1985

(4) Pavlov R., Eskenasi A., Mitkov R. - KATEST - testing system for administration of the motor vehicle driver's exam, Proceedings of Perscomp, Sofia, 1985 (in bulgarian)

(5) Mitkov R., Pavlov R., Eskenasi A. - Testing in Bulgaria with microcomputers, to be published

(6) Mitkov R. - Computerization of testing - situation and perspectives ,(to be published in bulgarian)

(7) Eskenasi A. - Application No.1 , Contract between the Institute of mathematics and the Sofia Council for popular education, Sofia, 1986 (unpublished)

((8) Lippey G.,(Ed.) - Computer-assisted test construction, Educational Technology Publications,USA,1974

(9) Vickers F. - Computer generated problem sets. A practical approach to computers in education, Educational Technology, 1973, 13(10), p.47-50

(10)Eskenasi A., Sabev V. - A method for a computer-assisted test construction, Serdica Bulgaricae mathematicae publicationes. Vol.11, 1985, p.54-58

(11) Hambleton R., Murray L. - Testing in the United States with computers, in Schlegel J.(Ed) : Bulletin of the inernational test commission and of the division of psychological assessment of the IAAP, No 20,1984

SOFTWARE FOR THE HOMECOMPUTER ROBOTRON Z 9001 - BASIC - CODES FOR SOLVING NUMERICAL PROBLEMS

Gerd Pönisch

The homecomputer robotron Z 9001 is produced since 36 months. It has an 8 bit processor running at 2.5 MHz clock rate and an user memory of 48 k bytes. A 4 k bytes ROM contains the operating system. The BASIC interpreter is available of a 10 k bytes ROM. It has a floating point arithmetic with a 3 bytes mantissa. The BASIC interpreter allows many string manipulations where 128 graphic signs can be used.
In the G.D.R. the homecomputer is used at the schools, high schools and universities as well as in scientific institutes and in the managment. Our department of numerical analysis get the order to compose some numerical algorithms for solving basis problems from linear algebra and nonlinear equations. For the choice of suitable algorithms we make a point of it that the algorithms have a favourable rounding error analysis and a good efficiency index.

However, the numerical part encloses only 10% of the statments of such a program. It is necessary to realize a comfortable dialog between user and computer. The program leads the user to the formulation of the desired task by questions or by menus, where a standard answer is suggested. In any phase the picture screen is shaped attractively. If the user gives an incorrect answer, after a jingle the user can answer fresh. All data used for the numerical computation are listed at the picture screen after the input for control. At this place it is possible to modify some of it in a simple way. This guaranties also the untrained user an easy work with our programs.

1. Linear algebra

At first we present three codes solving problems from linear algebra. The program LINGEN generates a LU decomposition for a nonsingular matrix A of n columns and n rows using a Gauss method with column pivoting and row equilibration. The LU factors are stored in the place of the matrix A, where the information about pivoting and equilibration are noticed in a vector of dimension n.

Using this LU decomposition of A the user can now solve the four tasks:

(i) Compute a numerical solution of the system $Ax = b$ of n linear equations.

(ii) Compute a numerical solution of the system $A^Tx = b$, where A^T denotes the transposed matrix of A.

(iii) Compute the inverse matrix A^{-1} of A

(iv) Compute the determinant det(A) of A

These four tasks can be performed repeatedly. Hence, the task (i) and the task (ii) can be solved for different right hand sides. Since a copy of A is stored the defect vector $d = A\tilde{x} - b$ is computed with higher accuracy for each numerical solution $\tilde{x}$ of $Ax = b$ and $A^Tx = b$, respectively. The user can estimate the goodness of $\tilde{x}$ by the maximum norm of d and can start an iterative correction of $\tilde{x}$. In this way the system $Ac = d$ is solved using the LU decomposition of A. The fresh numerical solution $\tilde{\tilde{x}}$ is given by $\tilde{\tilde{x}} = \tilde{x} - c$. Further iterative corrections of the numerical solutions are possible if the defect vector is not small enough. In praxis one or two corrections are be sufficient.

The inverse A^{-1} is computed by solving the n systems

$$Ax^k = e^k \quad , \quad k = 1,2,\ldots,n ,$$

where $e^k = (0,\ldots,0,1,0,\ldots,0)^T$ denotes the k-th unit vector of dimension n. The determinant of A is given by the product of the diagonal elements of the upper triangle matrix U. The code LINGEN needs a storage of $(8500 + 8n^2 + 12 \cdot n)$ bytes. Because of the accumulation of the rounding errors the dimension n should be not greater than 50.

If the matrix A is symmetric then the code LINSYM can be recommended. In this case a LDL^T decomposition of A is computed by the method suggested by Bunch and Parlett/1/. Note, that A has not to be positive definite for this method. The implemented algorithm works with symmetrical pivoting. The factor L is a lower triangle matrix with unit diagonal. The matrix D is block diagonal where the size of the symmetric blocks is 1×1 or 2×2. Because of the structure of L and D we need only one triangle matrix to store the factorization. The information about pivoting is keeped in one vector of dimension n.

Using the LDL^T decomposition of A we can solve the following tasks:

(i) Compute a numerical solution of the system $Ax = b$ of n linear equations.

(ii) Compute the determinant det(A) of A.

(iii) Give the eigenvalue characterization of A, i.e. the number of positive and negative eigenvalues of A.

These three tasks can be performed repeatedly. The task (i) can be solved for different right hand sides. After the computation of a numerical solution $\tilde{x}$ of $Ax = b$ the defect vector $d = A\tilde{x} - b$ is computed with higher accuracy. Therefore the information of the symmetric matrix A is doubled stored in an array of dimension $n \times n$ and in a n-dimensional vector. Analogously to the code LINGEN an iterative correction for $\tilde{x}$ is possible.

The determinant and the eigenvalue characterization of A is computed from D.

This code needs a storage of $(8900 + 4n^2 + 16n)$ bytes.

Because of the accumulation of the rounding errors the dimension n should be not greater than 50 .

If the matrix A has n columns and m($\geq$n) rows, we compute the least squares solution of the overdetermined system $Ax = b$ of m equations in n unknowns. The code LINREG generates a QR factorization of A using fast Givens rotations/2,3/. This implementation of Givens method is no more expensive than the well known Householder method. However, from the view point of rounding errors the fast Givens method is useful.

At first, we transform the rectangle matrix A by a finite sequence of plane rotations into a upper triangle matrix R :

$$R_o := A \; , \; D_{11} := \mathrm{diag}(d_1^{11}, d_2^{11}, \ldots, d_m^{11}) \; , \; d_k^{11} = 1 \; , \; k=1,\ldots,m$$

$$R_i := F_{im} \cdot F_{im-1} \cdot \ldots \cdot F_{ii+1} \cdot R_{i-1} \; , \quad i = 1,2,\ldots,n$$

$$F_{ij} := \begin{array}{cc}
\begin{array}{cccccccc} & & & i & & j & & \end{array} & \\
\left[\begin{array}{cccccccc}
1 & & & \vdots & & \vdots & & \\
 & 1 & & \vdots & & \vdots & & \\
 & & \ddots & \vdots & & \vdots & & \\
\cdots & \cdots & \cdots & 1 & \cdots & \beta & \cdots & \cdots \\
 & & & \vdots & \ddots & \vdots & & \\
\cdots & \cdots & \cdots & \gamma & \cdots & 1 & \cdots & \cdots \\
 & & & \vdots & & \vdots & \ddots & \\
 & & & \vdots & & \vdots & & 1
\end{array}\right] &
\begin{array}{c} \\ \\ \\ -i \\ \\ -j \\ \\ \\ \end{array}
\end{array}
\qquad
\begin{array}{l} i = 1,2,\ldots,n \; , \\ j = i+1,\ldots,m \end{array}$$

$$D_{ij} := \mathrm{diag}(d_1^{ij-1}, \ldots, d_i^{ij-1}/\tau, \ldots, d_j^{ij-1}/\tau, \ldots, d_m^{ij-1}) \; ,$$

$$\tau := 1 + \beta\gamma \; .$$

The coefficients β and γ are determined such that

$$(e^i)^T D_{ij} F_{ij} (F_{ij-1} \cdot \dots \cdot F_{ii+1} R_{i-1}) e^j = 0 ,$$

where e^l denotes the l-th coordinate unit vector. After $n(m - \frac{n+1}{2})$ elementar rotations the matrix A is transformed into $R = R_n$. By the same rotations we transform the right hand side b into $\overline{b} := (b_1^T, b_2^T)^T$ where b_1 is of dimension n. In the backsubstitution we have to solve the simple system $Rx = b_1$ of Dimension n.
The information which allow us to compute β and γ can be stored at the place of $(e^i)^T A e^j$. Hence we need only one additional vector in order to update the diagonal matrix D_{ij}. Our implementation works with pivoting. The information of it is noticed in a further vector.

This method can also used to solve a linear regression problem

$$g(x_1,\dots,x_n) = \sum_{j=1}^{m} (q(t_j, x_1,x_2,\dots,x_n) - y_j)^2$$

$$g(x_1,x_2,\dots,x_n) \longrightarrow \text{minimum} .$$

In this case the user has to define the linear ansatz function

$$q(t, x_1,x_2,\dots,x_n) = \sum_{i=1}^{n} x_i \cdot q_i(t) ,$$

where the functions $q_i : \mathbb{R} \longrightarrow \mathbb{R}$, $i=1,2,\dots,n$, are linear independent. After the input of $m (\geq n)$ measuring values $(t_1,y_1), (t_2,y_2), \dots, (t_m,y_m)$

the program generates the matrix A and the right hand side b in the form

$$A = \begin{bmatrix} q_1(t_1) & q_2(t_1) & \cdots & q_n(t_1) \\ q_1(t_2) & q_2(t_2) & \cdots & q_n(t_2) \\ \vdots & \vdots & & \vdots \\ q_1(t_m) & q_2(t_m) & \cdots & q_n(t_m) \end{bmatrix}, \quad b = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

By solving the overdetermined system $Ax = b$ by the QR technique mentioned above, the solution $x = (x_1, x_2, \ldots, x_n)^T$ of the least squares problem $g(x) \longrightarrow$ minimum is obtained. The computation of least squares solutions is more efficiently in this way than by solving the system $A^TAx = A^Tb$ which may be ill-conditioned too.

The measuring values (t_j, y_j), $j=1,2, \ldots ,m$, are also stored in a region of the user memory which is not administered by the BASIC-interpreter. Hence, it is possible to change the ansatz function q and start the program again in order to solve an other regression problem with the same or insignificantly modified measuring values. This is vary practicably, if the user does not know a appropriate ansatz function a priori.

The code needs a storage of (9000 + 4mn + 8m + 12n) bytes. This program is tested by many real life problems with success and can be recommended for small problems where n and m are not greater than 50.

2. Nonlinear problems

Four nonlinear problems can be solved by our software in time. The code FUNKNU computes a zero of a scalar function $f:\mathbb{R} \longrightarrow \mathbb{R}$ with enclosing techniques. For describing the function f the user has to define an user function. Then the graph of the function f can be plotted at the picture screen in a given interval. For finding a zero of f the user can choose between four algorithms which do not use the derivative f' of f . The standard method is a speded up regula falsi method(Pegasus algorithm/4/) with the R-convergence order 5 . An alternative method is an inverse interpolation method/5/. The bisection method can be used if it necessary to compute starting values for the local convergent methods. In order to compute multiple zeros efficiently a modified regula falsi method is applicated. It is possible to change the method during the computation. If the convergence is slowly then the iteration sequence converges to a multiple zero probably. In this case the program choose automatically the modified regula falsi . If the graph of the function is plotted the code find out good starting values for one zero in the given interval. If this zero is wanted the Pegasus method computes the zero in a few steps.

The code FUNKNU needs a storage of 8000 bytes.

If f is a polynomial the code POLYNU should be used. It computes all roots of a polynomial which can also have complex coefficients. The method of Nickel/6/ is implemented which computes automatically good starting values for the roots. The basis algorithm of it is Newton's method in the complex space. Because of the so-called pivoting strategy the method works stable and computes roots with high accuracy.

After the computation the roots are plotted at the picture screen into a diagram. The code needs a storage of (6500 + 20n) bytes, where n is the degree of the polynomial.

For solving a system of nonlinear equations the damped Gauss-Newton method is implemented in the code NLREG. This program computes a least squares solution of the system

$$\begin{aligned} F_1(x) &= 0 \\ F_2(x) &= 0 \quad , \quad F_j : \mathbb{R}^n \longrightarrow \mathbb{R}^1 \ , \ j=1,2,\dots,m \ , \\ &\vdots \\ F_m(x) &= 0 \end{aligned}$$

of m nonlinear equations in n variables $x = (x_1, \dots, x_n)^T$. Starting from a initial guess x^0 the new iterate is computed by

$$x^{k+1} := x^k - \lambda_k p^k \quad , \ k=0,1,2,\dots \ ,$$

where the descent direction p^k is choosen as the Gauss-Newton direction

$$p^k := (F'(x^k)^T F'(x^k))^{-1} F'(x^k)^T F(x^k) .$$

The step size parameter λ_k is evaluated by the algorithm of Goldstein and Armijo/7/. The direction p^k is efficiently computed by solving the corresponding linear least squares problem using fast Givens rotations. Instead of the explicit use of $F'(x^k)$ we work with consistent approximations. Hence,the user have only to write a subprogram for computing the actuell function values $F(.) = (F_1(.),F_2(.),\dots,F_m(.))^T$. In the case $m = n$ the code solves a system of n equations by the damped Newton method.

In particular, it is easy possible to solve nonlinear regression problems with this Gauss-Newton method. In this case the user have to define an ansatz function

$$y = q(t, x_1,x_2,\dots,x_n)$$

with the free parameters $x_1,x_2,\dots,x_n$. After the input of m ($\geq$n) measuring values $(t_1,y_1),\dots,(t_m,y_m)$ the programm generates the corresponding overdetermined system

$$\begin{array}{lcl} F_1(x) := q(t_1 , x_1,x_2,\dots,x_n) - y_1 & = & 0 \\ \vdots \qquad\qquad \vdots \qquad\qquad\qquad \vdots & & \vdots \\ F_m(x) := q(t_m , x_1,x_2,\dots,x_n) - y_m & = & 0 \quad , \end{array}$$

which can be solved by the Gauss-Newton method in an efficient way. Analogously to the code LINREG the measuring data are stored in a special region of the user memory. Hence we can easy use these values for

many runs.
The code NLREG needs a storage of (8500 + 4mn + 12 m + 16n) bytes. Analysing a number of examples we have to remark, that this code is suitable to solve weakly nonlinear problems. Because of the single precision arithmetic the implemented algorithm may be fail at strong nonlinear examples.

All these codes are written in the technique of subprograms. Thus, they can easy adapted for personel computes which work with a BASIC compiler or interpreter. The codes created together with a team of students belong to the software offer of the producer of the homecomputer Z 9001 .

References

/ 1 / Bunch, J.R.; Kaufmann, L.; Parlett, B.N.:
Decomposition of a symmetric matrix.
Numer. Math. 27/1 (1976), pp. 95 - 1o9

/ 2 / Stewart, G,W.: The economical storage of
plane rotations.
Numer. Math. 25/2 (1976), pp. 137 - 139

/ 3 / Schwetlick, H.; Kielbasinski, A.: Numerische Verfahren der Linearen Algebra.
Verlag d. Deut. Wiss., Berlin, in press

/4/ Anderson, N.; Bjorck, A.: A new high order method of regula falsi type for computing a root of an equation.
BIT 13 (1973), pp. 253 - 264

/5/ Ostrowski, A.M.: Solution of equations and systems of equations. 2nd. ed.,
Academic Press, New York 1966

/6/ Nickel, K.: Die numerische Berechnung der Wurzeln eines Polynoms.
Numer. Math. 9/1 (1966), pp. 80 - 89

/7/ Schwetlick, H.: Numerische Lösung nichtlinearer Gleichungen.
Verlag d. Deut. Wiss., Berlin, 1979 .

Gerd Pönisch
Technische Universität Dresden
Sektion Mathematik

Mommsenstr. 13
Dresden
8027

German Democratic Republic

An Approach to Programming by Means of Equations: Transformation Programs and an Interpreter for Such Programs

A. Radensky, M. Todorova

The paper presents an approach to programming by means of equations, allowing the integration of the functional and logical style of programming. The aim of this approach is to develop a program language and an interpreter for it. Some basic characteristics of the language and the interpreter are given.

1. Programming by means of equations in a logical style

Programming by equations is a particular type of functional programming. In it, a function is defined through a system of equations (an equation program), each equation of which gives various parts of the definition, for example:

/1/ length (nil) = 0

/2/ length (cons (X,Y)) = length (Y) + 1

The theory of computation through systems, described by equations, has been developed by M. O' Donnell (1). O'Donnell considers the programs with equations to be a set of equations of the A = B type, where A and B are terms or Σ-trees. The equations are orientated in the sense that B can be substituted for A, while A cannot be substituted for B.

Thus, both logical programs and programs with equations are descriptions of some properties. In logical programs, properties can be described by Horn clauses, while in programs with equations they are described by means of equations. On the other hand, both logical programs and programs with equations can be interpreted as descriptions of computations. Thus, both these types of programs have two kinds of semantics: conventional mathematical semantics, and procedure semantics.

The equation interpreter takes a program with equations and an expression as an input, and yields the value of the expression as an output.

The logical interpreter takes a logical program and a question as an input, and it obtains a corresponding answer.

There is, however, an essential difference between the equation and the logical interpreters. Questions put to logical interpreters may contain variables. The values of these variables are determined by the logical interpreter. The expressions that are estimated by equation interpreters cannot contain free variables. At the same time, substitutions of variables are not allowed in the course of computations.

The present approach to programming by equations eliminates this difference. It is based on the simple fact that unification can be used in the course of computations of an arbitrary functional term, and not only of predicates. Thus, it becomes possible to estimate terms with free variables by suitable substitutions for the variables. The method by which this estimation is done is similar to the way Prolog-interpreters respond to questions with variables.

Computation of expressions with (free) variables are referred to as the logical style of estimation. The equation language described in this paper performs such kind of estimations. The logical style of estimating will be illustrated by the following example.

Example: Let us compute the expression equal (length(X),1) by means of the following equation program

```
T1    length(◊) = 0
T2    length(XS) = length(S) + 1
T3    equal(0,0) = true
T4    equal(M+1,0) = false
T5    equal(0,N+1) = false
T6    equal(M+1,N+1) = equal(M,N)
```

At each step of assessing, the leftmost and the outermost substitution is performed. If two or more equations are applicable in one and the same step, assessing continues in two or more independent branches.

T1 and T2 are applicable to equal(length(X),1). Substitution X = $\Diamond$ transforms the expression equal (length(X),1) into expression equal(length($\Diamond$),1); while X = X_1S_1 transforms equal(length(X),1) into equal(length(X_1S_1),1). By means of equation T1, the expression equal(length($\Diamond$),1) is reduced to equal(0,1); and by T5, the latter expression is reduced to "false". The logical style of estimating the expression equal(length(X),1) is presented by the following diagram:

$$\begin{tikzcd}
 & \text{equal(length(X),1)} \arrow[dl, "X=\Diamond"'] \arrow[dddd, "X=X_1S_1"] \\
\text{equal(length(}\Diamond\text{),1)},X=\Diamond \arrow[d, "T1"] & \\
\text{equal(0,1)},X=\Diamond \arrow[d, "T5"] & \\
\text{false}, X=\Diamond & \\
 & \text{equal(length(}X_1S_1\text{),1)},X=X_1S_1 \arrow[d, "T2"] \\
 & \text{equal(length(}S_1\text{)+1,1)},X=X_1S_1 \arrow[d, "T6"] \\
 & \text{equal(length(}S_1\text{),0)},X=X_1S_1 \arrow[dl, "S_1=\Diamond"'] \arrow[ddddl, "S_1=X_2S_2"] \\
\text{equal(length(}\Diamond\text{),0)},X=X_1\Diamond \arrow[d, "T1"] & \\
\text{equal(0,0)},X=X_1\Diamond \arrow[d, "T3"] & \\
\text{true},X=X_1\Diamond & \\
\text{equal(length(}X_2S_2\text{),0)}, X=X_1X_2S_2 \arrow[d, "T2"] & \\
\text{equal(length(}S_2\text{)+1,0)}, X=X_1X_2S_2 \arrow[d, "T4"] & \\
\text{false}, X=X_1X_2S_2 &
\end{tikzcd}$$

Therefore, the result of the logical estimation of the expression equal(length(X),1) is:

false if $X = \Diamond$
true if $X = X_1 \Diamond$
false if $X = X_1 X_2 S_2$

Thus, a new and a more valuable interpretation is introduced concerning the conventional equation programs.

This approach has been introduced in (2), and theoretically formulated in (3). Reference (3) also proves that conventional equation languages produce logical style interpreters, without changing traditional semantics.

2. Generalized Transformers

On the basis of the approach described above, the W program language has been created. Programs written in this language will be referred to as generalized transformers or transformer programs.

The W language is a language for programming by means of equations, and it is generalized in two directions.

The first generalization is sintactic. Conventional equation interpreters act as transformers of functional terms. A more general approach has been realized in the W language: generalized transformers describe transformations of strings, and not only of functional terms. The structure of these strings is described in the generalized transformer by means of context-free grammar.

The second generalization is semantic. The existing equation interpreters estimate only terms which do not contain variables. In the W language, a "logical" estimation of terms is possible.

Generalized transformers are single-argument functions consisting of three parts: description of data types, description of variables, and transformations. Data types, processed by the generalized transformers, are described by a version of the methods of Becus-Naur. Transformations are rules according to which the argument of the generalized transformer is transformed by subsequent steps into a value of the generalized transformer. Transformations can contain variables which are described in the section on variables. The argument of the generalized transformer does not participate explicitly in its definition.

The generalized transformer LENGTH defines the function length , defined above.

```
H1  TRANSFORMER  LENGTH:<EXPR>;
H2  TYPE
H3  <EXPR>::=<NAT>;
H4  <NAT>::='LENGTH('<LIST>')';
H5  <NAT>::=<NAT>'+'<NAT>;
H6  <NAT>::='0';
H7  <NAT>::='1';
H8  <LIST>::='CONS('<LIST>','<LIST>')':
H9  <LIST>::='NIL';
H10 VAR
H11   <X>: <LIST>;
H12   <Y>: <LIST>;
H13   <Z>: <LIST>;
H14 BEGIN
H15  'LENGTH(NIL)'=='0';
H16  'LENGTH(CONS('<X>','<Y>'))'=='LENGTH('<Y>')
     +1';
H17 END
```

Each line of this program is provided with a number for the sake of clarity. The first line contains the title of the generalized transformer. Lines H2 through H9 cover the section for type description; H10 through H13 cover the section for variable description; and H14 throught H17 - the section for transformations.

By applying this generalized transformer to the expression 'LENGTH('<Z>')', we get:

```
( 0   if   Z = NIL,
  0+1   if   Z = CONS(X,NIL),
  0+1+1   if   Z = CONS(X,CONS(X,NIL)),
  0+1+1+1   if   Z = CONS(X,CONS(X,CONS(X,NIL))),
  ..........)
```

Here follows a brief description of the syntax of the W language. Symbols underlined by " ~ " are symbols of the W language.

```
<description of generalized transformer>::=
            <title> <block>
<title>::=TRANSFORMER <name> : <main type> ;
<block> ::=  <section of types>
             <section of variables>
             <section of transformations>
<section of types> ::=  type
                        .<definition of type>
                       {;<definition of type>}
<type> ::=  <primitive type> | <composition type> |
            <defined type>
<main type> ::= <type>
<primitive type> ::=   ' <string> '
<composition type> ::= <P - composition> |
                       <N - composition>
```

```
<P - composition> ::= <primitive type>|
        <primitive type> <name of type >|
        <primitive type > <N - composition>
<N - composition > ::=  <nyme of type><name of type>|
        <name of type> <primitive type> |
        <name of type> <P - composition >|
        <name of type> <N - composition>
<name of type>  ::=  <<name>>
<definition of type> ::=
      <defined type > ::= <defining type>
                          {!<defining type>}
<defined type> ::=    <<name>>
<defining type > ::=  <type>
<section of variables>  ::=  <blank>|
               <description of variables >
             {;<description of variables >}
<description of variables>  ::= < variable >
                         {, < variable >} :  < type>
<variable > ::= <<name >>
<section of transformations> ::= <blank> |
                BEGIN
                  <transformation>
                {;<transformation >}
                END
<transformation > ::=  <blank > |
           <sample > ==  <sample>
<sample > ::== <C - sample > |< V - sample>
<C - sample> ::=  ' <string>' | '<string>'<V -
                                       sample >
<V - sample> ::= <variable>
```

<variable> <V - sample>|

<variable> <C - sample>

"Name" in the W language means a sequence of letters, numbers and underscores "_", which sequence begins with a letter.

If B is a defining type which participates in the definition of type A, this will be marked by $A \longrightarrow B$. Let X and Y be types, and when $i = 1,2$, Z_i will either mean a type, or blank . Let the notation of the type of X be $Z_1 A Z_2$ and $A \longrightarrow B$. If Y is obtained as a concatenation of Z_1 , B and Z_2 , then we write $X \Longrightarrow Y$.

We say that *Y is a subtype of X*, and we write $X \Longrightarrow_{*} Y$ if either Y = X, or there exist such types X_0 , X_1 , ..., X_r , that $X_0 = X$, $X_r = Y$, and $X_i \Longrightarrow X_{i+1}$, for each $i = 0, \dots, r - 1$.

If Y is a subtype of X, *X is a supertype of Y*.

The type description section assigns a grammar that has the $A \longrightarrow B$ relations between the types as its rules, and the main type as its axiom. The relation of direct derivation in this grammar is similar to the relation $A \Longrightarrow B$. It is necessary that the grammar of the types should be unambiguous.

Let us have preset values of the variables. The value of the sample is a string which is obtained when strings and values of variables are joined in the line on which they appear in the sample. By substituting the notations of the variable types for the names of all variables in the sample, we obtain a type which is referred to as a <u>type, generated by the sample</u>. <u>Type of the sample</u> is each primitive or defined supertype of the type generated by the sample.

<u>Type of transformation</u> is each type of the sample which is to the left of the transformation.

In order to create more effective programs, it is recommended that the right side of the transformation should be a sample which is of the same type as the sample in the left side.

Suppose we have identical types generated by the left sides of two transformations. If the transformations themselves are not identical, then they are inconsistent. Transformations are considered to be consistent if they are not inconsistent.

It is desirable to consider only such generalized transformers that have consistent transformations.

Expressions in the W language are also referred to as T-strings.

⟨T-string⟩ ::= <sample> {if <assumption>}
¤ {if <assumption>} ,

where

<assumption> ::= (<variable> = <sample>
{;<variable> =<sample>}) |
<blank> .

The generalized transformer is applied to T-strings with a blank assumption.

<application of generalized transformer to T-string>
::= <name> (<T-string>).

The result of the application of a generalized transformer to a T-string is the list of T-strings:

<list of T-strings> ::= (<sequence of T-strings>)

<sequence of T-strings> ::= <T-string> |

<T-string>, <sequence of T-strings>

We shall give a brief description of the semantics of the W language. First, we shall define the way a transformation is applied to a T-string.

The sample of each T-string may be considered to be a concatenation of three samples which will be referred to as head, body and tail. A set of such representations exists for each sample.

We shall discuss the way in which transformation U == V is applied to T-string t if σ.

Suppose that with a suitable selection of values for variables in the U sample, the value of U cincides with a body of t. Then we say that U== V is P-applicable to T-string t if σ . The assumption to the left of the "=" signs, in which there are variables that are present in the sample of the transformation while on the right there are substrings of t, will be referred to as P-assumption defined by transformation U == V.

Suppose that with a suitable selection of values for variables of body t, the value of the body coincides with sample U of transformation U == V. Then we say that U == V is S-applicable to T-string t if σ . The assupmtion in the left sides of which there are variables pertaining to any T-string while on the right there are substrings of U, will be referred to as S-assumption defined by transformation U == V.

Suppose that with a suitable selection of values for variables of a body in T-string t if σ, as well as values for variables of U, the values of the body coincide with the value of U. Then we say that transformation U == V is PS-applicable to t if σ.

Therefore, if U == V is PS-applicable to t if σ, then it is at the same time P-applicable and S-applicable to this string.

Transformation U == V is applicable to T-string t if σ, in case it is either P-applicable, or S-applicable, or PS-applicable. In such case the transformation and the body of the T-string have at least one common type which is referred to as a type of application. In addition to the above-stated definition concerning the applicability of a transformation to a T-string, a composition type is also required which is defined by the head of the T-string, the type of application, and the tail of the T-string, and it should be a subtype of the main type.

Transformation U == V is applied to T-string t if σ, depending on the type of its applicability. If the transformation is P-applicable to t if σ, it is applied to it in the following way: the value of V is found by the P-assumption defined by the transformation, then this value is substituted for the body of the T-string. σ is not changed.

If U == V is S-applicable to t if σ, it is applied to it in the following way: the value of t if σ is found by S-assumption δ, defined by the transformation, i.e. $\delta(t)$ *if* $\delta \circ \sigma$. In this T-string, we substitute sample V for the value of the body.

If U == V is PS-applicable to t if σ, it is applied to it in the following way: the value of t if σ is found by S-assumption δ, defined by the transformation, i.e. $\delta(t)$ *if* $\delta \circ \sigma$. In this T-string we substitute the value of V by P-assumption defined by transformation U == V, for the value of the body by means of δ.

A set of transformations, applicable to a T-string, is referred to a set of transformations equivalent with respect to applicability, if the bodies, the heads and the types of application of all transformations of the set are the same. A set of transformations equivalent with respect to applicability is applied to a T-string in the following way: each transformation is applied to the T-string depending on its type of applicability. Thus a list of T-strings is obtained, which is the result of the application.

Suppose two sets of transformations are given, equivalent with respect to applicability, that are applicable to one and the same T-string. For each one of the two sets there exists any kind of a head, of a body, and a type of application. We say that the first set is applicable before the second, if one of the following conditions is satisfied:

1. The head in the case of the first set is shorter than in the case of the second set.

2. The two heads are equal, but the body in the case of the first set is longer than the body in the case of the second.

3. Heads and bodies of the two sets are the same but the types of application are different, and the type of application of the second set is a subtype of the type of application of the first set.

The computation of a generalized transformer with a preset argument is perfomed in subsequent steps. A check is made at each step whether the argument is of the main type. If it is not, the value of the generalized transformer is indefined (i.e. the value is ¤). If the argument is of the main type, those transformations are sought that are applicable to the argument. If there is not such a transformation, computation for this argument end up. Otherwise, all applicable transformations are divided into sets of transformations, equivalent with respect to applicability. Among all applicable transformations, this one is selected that is applicable before the others. The set thus chosen is applied to the argument. This way, a list of T-strings is obtained, each element of which is assumed to be an argument of the generalized transformer.

3. Experimental System of Programming by Means of Generalized Transformers

Strings pertaining to a context-free language may be represented by functional terms. On this basis, generalized transformers may be converted into conventional equation programs in which functional terms should be used, but which compute terms with varia bles. Such programming systems are called generalized systems for programming with equations. This allows the setting-up of an experimental system of programming by generalized transformers of the semi-interpreting type.

The experimental system consists of:

- a preparatory part including a lexical analyzer, syntax analyzer, and TF-converter;
- an interpreter which computes the term corresponding to the input T-string by the generalized equation program, obtained by the generalized transformer;
- an FT-converter.

This organization is illustrated in Fig.1.

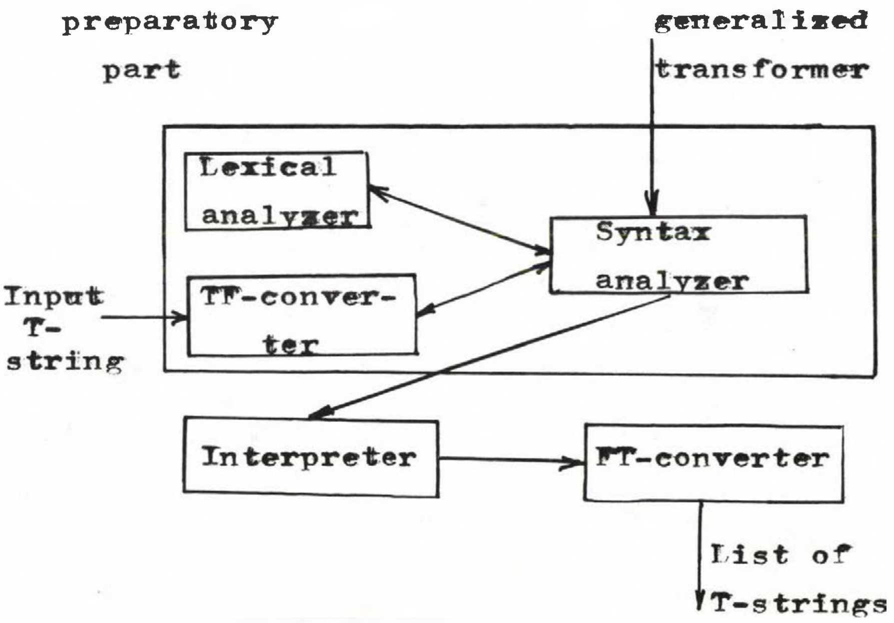


Fig.1

References

1. O'Donnell M., Computing in Systems Described by Equations, LNCS, vol.58, Springer-Verlag (1977)
2. Radensky A., Functional Programming in the Style of Logical Programming, C.R. Acad. Bulg. Sci. 37 (1984), p.741-744
3. Todorova M., On the Confluency of Generalized Subtree Replacement Systems, C.R. Acad. Bulg. Sci. (to be printed)

COMPUTER - AIDED DATABASE MANAGEMENT SYSTEM IN the HERNAD "MARCIUS 15. MGTSZ" AGRICULTURAL CO-OPERATIVE

A Case Study

REMZSO GABOR
TECHNICAL UNIVERSITY OF BUDAPEST
Computer Centre

1. The most important activities in the co-operative

The HERNAD "Marcius 15. MGTSZ" Agricultural Co-operative is the cooperator of the *HUNNIAHIBRID* production association which was called to existence for the production of broiler-chicken. In this field of activity the Cooperative has a close business link with the Dutch *EUROHIBRID* company. In the process of the broiler production small estates have a high importance as most of the breeding takes place there. The HUNNIAHIBRID is a union of many Hungarian agricultural co-operatives working on the poultry breeding area. The most important activities in the co-operative are:

animal breeding and processing
plant growing
fodder processing
management farming plots
complementary industrial activities
food processing
consulting members of the HUNNIAHIBRID

With respect to its monetary value the most important is the animal breeding (and poultry production, respectively) branch.

2.Computers in the co-operative

In the HERNAD "Marcius 15. MGTSZ" Agricultural Co-operative there are the following computer hardware systems:

IBM SERIES/1
INTEL 80 Data Collection System
VIDEOTON EC 10/M (VIDEOPLEX)
IBM PC/XT

In the centre of the computer system is the IBM SERIES/1 computer. This computer has the following hardware configuration:

PROCESSOR IBM 4955-F00 256KByte
Timers
Floating Point
Communication Indication Panel
Programmable Communication Features

I/O Expansion Unit IBM 4959-A00

Diskette Magazine Unit 27.8 MByte

Disk Subsystem 28 MByte

Display terminals IBM 4978-1 4 pcs

Matrix printers IBM 4974-1 3 pcs

Teletype Displays ORION ADP 2000 18 pcs

TTY lines

THE CONFIGURATION OF THE EC-10/M:

128 KByte CPU

Magnetic tape units (2*)

Disk 20 MByte

Punched card reader

Line printer

THE CONFIGURATION OF THE IBM PC XTS:

CPU 256 KByte

Winchester disk capacity 10 MByte

Floppy disk

Matrix printer

IBM 8087 Math. Coprocessor

3.Systems on the SERIES/1

In the IBM SERIES/1 the following systems were developed (*HERNAD-INFO*) :

Collection of poultry production branch data

Book-Keeping System

Farm Accounting System

Accounting system

Marketing system

Stock Management

Optimal Control of the Fodder Plant

Production Information (daily,weekly, montly,yearly)

Summarized information of the HUNNIAHIBRID system.

4.THE WORK OF THE VIDEOPLEX

On the VIDEOPLEX there is a data preparing system for the other computers.

The large capacity line printer is connected to the IBM SERIES/1 too.

5.THE WORK OF THE INTEL 80 SYSTEM

This system is the part of the LINDHOLST Chicken Slaughtering Equipment. The collected data of this computer system is sending to the IBM SERIES/1.

6.ACTIVITIES IN THE IBM PC/XT COMPUTERS

We proposed to make a Consulting system for the HUNNIAHIBRID union, the name is ***HUNNIA.***

This system connecting with the other systems serves the following data:

information about partners and contracts;

information for poultry breeding technology;

statistical data of the HUNNIAHIBRID system;

information for the optimal production of *animals;*

collected information of the HUNNIAHIBRID;

trace the egg production;

trace the hatching and the fodder processing data.

One part of these systems is developed and used, the other part (HUNNIA) is under development .

REFERENCES

1.Date,C.J., An Introduction to Database Systems
Addison-Wesley Publishing Company
1977.

2.Benasteau,D., Comment creer votre propre banque de donnees,et la reussir
L'Usine Nouvelle
No 21. 1985.

3.Mill,J., An executive tool or toy?
Computing the Magazine
No 4. 1985.

OFFICE AUTOMATION and DATA PROCESSING

Remzso Tibor

Computer and Automation Institute of the
Hungarian Academy
of Sciences

INTRODUCTION

When first introduced, the microcomputer was a revolutionary development, bringing quick and convenient computing right down to the desktop level. As a tool of business or office, the microcomputer had some serious deficiencies.

In effect, it offered only half a solution to the problem of office automation.

The other half is the local area network - the indispensable partner of the microcomputers in the office.

Most of this lecture deals with how set up and organize this powerful combination: the microcomputer technique and the local area network.

WHAT IS A LOCAL AREA NETWORK?

First let's examine what a local area network is and why it is essential and useful in using microcomputers in the office.

A local area network is a communications system much like a telephone system. Any connected device can use the network to send and receive information. For the time being, that information on IBM PC networks is almost exclusively data, although the technology is now available for carrying voice and video signals also.

As the name implies, a local area network is used to cover relatively short distances. Usually a local area network will be limited to a department or a single building. The most common network for IBM PC-like users contains from 3 to 10 PCs, various data storage devices, printers and other specialized peripherals (modems, digitizers, etc.)

An important charcteristic of local area networks is speed. They deliver data very fast. A person passing and receiving data over a local area network ideally experiences the same rapid response time as if the data were coming from a local machine rather than from some place out on the network. To get this kind of response time, most local area networks operate at 1 to 10 Mbits per second speed.

Besides being fast, local area networks must be both adaptable and reliable. They have a flexible architecture that permits PC workstations to be located wherever they are needed. PCs or peripherals should easily be added or removed from the system without any extended interruption in the operation of the network.

One of the major benefits of a stand-alone PC was that if it breaks down or malfunction in some way, the effects were limited. The rest of the office work wasn't interrupted. Likewise, when personal computers are linked into a local area network, the system should retain this kind of reliability. The failure of a single PC should not cause the entire network to shut down.

A local area network is a micro-to-micro network with distributed intelligence. The personal computers attached to the network can use the computing power of other intelligent devices - as in a host-to-terminal network - but the personal computers can also use their own computing power.

COMPONENTS OF THE LOCAL AREA NETWORK

A local area network is a system made from building blocks that can be added and shaped as needed.

One of these components is the cable. Each device on the local area network is attached to a transmission cable usually a coaxial cable - so that messages can be sent from one device to another. For the interface, a circuit board is placed between the cable and the PCs and peripherals. This board is logically called the network interface card.

Central mass storage is provided in the form of a hard disk (e.g. Winchester disk) that contains files and programs which are shared by people using the network.

A local area network is a multiuser system because more than one person at a time can send requests to a single microprocessor. The shared machine, together with a software program, handles the requests and distributes the network resources, such as data files and printer time. Both the machine and its software are referred to jointly as the server.

PCs on the local area network continue to use their own local operating systems (e.g. PCDOS, MSDOS). Since these operating systems make no provision for using a network, appropriate software must be added. This software is called the network operating system.(E.g. I-NET, TRANS-NET, PCNET, 3COM).

ARGUMENTS FOR AND AGAINST NETWORKING

If we want to make a network, we must examine the features that networking brings to the office and weigh their advantages against the cost of networking.

Sharing expensive peripherals is often promoted as the primary reason to network. It is right in our countries, but isn't in the western countries.

Then we come to the question of sharing itself. Do most office users really want to share? A large part of the appeal and acceptance of the personal computer has been that it wasn't shared, that it was indeed available for personal use. Waiting in a queue to get computing time on a mainframe or to have a document printed is just not something that most people relish.

Sharing also raises another possible disadvantage. When three, four or ten people start using a large and fast hard disk, its speed can quickly go back to that of a local floppy drive.

These are very important considerations, but only in the one part. When viewied as a system, networking has some powerful arguments in its favor. Organizations with multiple PCs should network them for the following reasons.

a./ Sharing of peripherals reduces their cost per-user. Often a higher-quality peripheral can be justified as a shared resource, with the result that speed and quality are improved and MTBF is increased (Mean-Time-Between--Failures). When a device fails in the local area network, another one is ready to fill the void while repairs are being made.

b./ Better response time can be achieved through networking. If properly implemented, a PC local area network will be more efficient than stand-alone computers or networked terminals.

c./ The peripherals attached to a local area network tend to be faster than those dedicated to stand-alone PCs. The cable speed of all the local area networks far exceeds the speed capability of the PC with its 8088 microprocessor. For many applications the PC is the bottleneck. Using local area networks you may not able to speed up the PC itself, but you can speed up the results, because of a local area network is a multiprocessor-like system.

d./ Departments, offices are all organizations, which imply interaction and teamwork. Without a local area network the personal computers have been a powerful but isolated devices.Theirs output have been difficult to integrate into the organization mainstream . Networking is a communications mechanism that ties the isolated PC into the organization.

e./ Flexibility is a distinct advantage of a local area network. Microcomputers can use another computer's processing power instead of their own. Peripherals can be shared or dedicated. Finally, local area network users have options that can be tailored to achieve the right balance of performance and cost efficiency for the needs of a particular network.

THE LOCAL AREA NETWORKS AS COMMUNICATIONS DEVICES

The local area networks are communications devices. Letters, messages, memos and files can be sent from one PC to another. With this, a local area network can be connected to wide area networks through gateways . So that users can communicate with other local area networks, data base services, and remote PCs, too.

Communicating and sharing information have a side benefit that may actually be the best inducement of all for networking: a network promotes an organized computing environment. In many business the personal computer can be a disruptive force. It may causes difficulties, because everyone does things differently. The text files generated by one user are not usable by another because each person has a different word processor. Formats of documents tend to vary from machine to machine.

If information generated on a personal computer must be read by other machines or stored on a mainframe, much of the work may need to be redone to move it to a new system. The network with its shared central storage and channels of communication, requires user cooperation, which results in better organization and continuity of effort.

DATA SAFETY AND SECURITY

A local area network permits distributed processing and central storage. Distributed processing is a performance enhancer, but central storage is crucial to data control. On a network, data can be protected through a supervisor-administered backup system. Access to data can be limited and monitored with multiuser protection schemes (e.g. user passwords) , which are available on most networks.

Microcomputer local area networks are capable of providing a microcomputer environment with many of the security and data integrity protections common to multiuser terminal networks. At the same time, micro local area networks offer special controls that are necessary to handle intelligent workstations.

PROBLEMS CAUSED BY THE NETWORK

The local area networks can be a source of problems. that are specifically network-related. One type of these problems is the accidental loss of data when two or more people share the same data at the same time. A later modification of a file can overwrite the work of an earlier user. Any changes or additions made by the first user - who first stored the file - are lost.

A local area network's software must therefore be able to protect against the danger of accidental loss of data. There are simply and more sophisticated solutions for this problem.

Though networks prevent some security problems, they may create others. Someone can tap into the network by entering the cable at an accessible point. This may be dangerous.

REFERENCES

1. Flanagan,P., Need Information? You name it, databases supply it.

   Office Administration and Automation
   1983. No.4. pp. 42-48.

2. Hwang,K., Fu,K., Integrated computer architectures for image processing and database management

   Computer
   1983. No. 1. pp. 51-60.

3. Netravali,A.N., Bowen, E.G., A picture browsing system

   IEEE Transactions on Communications
   1981. No. 12. pp. 968-1976.

DEDUCTIVE BASIS OF RELATIONS - NEW EFFECTIVE NORMAL FORM FOR THE DESIGN OF RELATIONAL DATA BASES

B. Thalheim

Technische Universität, Mathematik, Dresden, GDR

The "normalization" of relations is one of the most important tools for database design. The concept of special kinds of dependencies has been proved to be useful in the design and analysis of relational databases /4,5/. By using this concept, a new ("deductive") normal form of relational databases is defined. This deductive normal form is better than other known normal forms in the most cases. By using special tuple-generating dependencies as deduction rules we get the entry relation from its deductive basis. During the query phase, the rules are used to generate all posible derivations of facts and thereby make them again explicit in the database. But from recursive deduction rules arises the termination problem when the rules are used since potentially they may lead to infinite derivation paths. However, in the case of single decomposition dependencies as deduction rules or in the case of classes of decomposition dependencies generated by a single decomposition dependency it is possible to find a termination condition which cuts potentially infinite derivation paths. Therefore in this paper a conditions for classes of decomposition dependencies to be generated by single decomposition dependencies is given moreover.

1. Basic notions and problems

We assume some familiarity with the relational model. In this section we define the elements of the model used in the paper.

Attributes or column names are symbols from a given finite set $U = \{A_1, A_2, \ldots, A_n)\}$ with a given fixed natural number n. We assume that with each attribute A there is associated a set, called its domain. Since in this paper are discussed only dependencies and deductive basis, w.l.o.g., we assume that there is one domain G.

A relational data structure (on U) is a system $<G , R>$, where R is a finite subset of G^U called relation.

For a subset X of U , a relation R and a tuple r of R we denote by $r(X)$ the restriction of r to X .

A special Horn-formulae

$P(x_{11}, \ldots, x_{1n}) \& \ldots \& P(x_{m1}, \ldots, x_{mn}) \longrightarrow P(x_{01}, \ldots, x_{on})$

is called strong tuple-generating dependency (TGD) if $x_{ij} = x_{kl}$ then $j = l$ $(0 \leq i,k \leq m,\ 1 \leq j,l \leq n)$ and if for all x_{oj} there is a k , $1 \leq k \leq m$, with

$x_{oj} = x_{kj}$,

and is called decomposition dependency (DD) if moreover for all i , j $(1 \leq i < j \leq m)$ and k $(1 \leq k \leq n)$ from $x_{ij} = x_{kj}$ follows $x_{ij} = x_{oj}$

(no hidden conditions in premise).

A join dependency (JD) is a cover $(X_1,\ldots,X_k)$ of U . A DD
$P(x_{11},\ldots,x_{1n}) \& \ldots \& P(x_{m1},\ldots,x_{mn}) \longrightarrow P(x_{o1},\ldots,x_{on})$
is equivalent to the JD $(X_1,\ldots,X_m)$
with $X_i = \{ A_j \;/\; x_{ij} = x_{oj} \}$.

A TGD means that if some tuples, fulfilling certain conditions, exist in the relation then another tuple must also exist in the relation.

Given a set M of TGD, a set G , a relation R on U and a set of interpretations $I(M) = \{ Pr^1 \& \ldots Pr^k \longrightarrow Pr \}$ of M in G . Then we define

$$/R/_{M,0} = R \;,$$
$$/R/_{M,i+1} = /R/_{M,i} \cup$$
$$\{ r \in G^U \;/\; r^1,\ldots,r^k \in /R/_{M,i} \;,\; Pr^1 \& .. \& Pr^k \longrightarrow Pr \in I(M) \}$$
$$/R/_M = \bigcup_{i=0}^{\infty} /R/_{M,i} \quad .$$

The set $/R/_M$ is called the M-closure of R .

<u>Corollary 1</u>. Given a finite relation R and a set of TGD M , there is a natural number i such that $/R/_M = /R/_{M,i}$.

A relation $R \subseteq G^U$ is called M-closed if holds $R = /R/_M$. A set of TGD M is true in R if holds $R = /R/_M$. A TGD β follows from a set M if all M-closed relations R are $\{\beta\}$-closed ($M \models \beta$).

Given a set of TGD M and a M-closed relation R . A subset R' of R is called M-deductive subset if holds $/R'/_M = R$. A M-deductive subset R' of R is called **M-deductive basis of R** if there is no M-deductive subset R'' of R with $R'' \underset{\neq}{\subset} R'$.

Given a relation R . Let be M_R the set of all TGD β with $/R/_{\{\beta\}} = R$. A M_R-deductive basis of R is called **deductive basis of R** .

<u>Example 1</u>. Given $G = \{0,1\}$, n=3, $U = \{1,2,3\}$, $\beta = P(x_1,x_2',x_3)\&P(x_1,x_2,x_3') \longrightarrow P(x_1,x_2,x_3)$ and $R = \{(0,0,0),(0,0,1),(0,1,0),(0,1,1),(1,0,0)\}$. The subsets $R' = \{(0,1,0),(0,0,1),(1,0,0)\}$ and $R'' = \{(0,0,0),(0,1,1),(1,0,0)\}$ are $\{\beta\}$-deductive bases of R .

There are two main problems.

1. Given a deductive basis R of a relation $/R/_M$. How many steps are needed for construction of the M-closure of R ? For a given set M are there bounds for construction of M-closure of relations ?
2. Given R and M . How to construct a M-deductive basis of R ?

For the second problem there are some algorithms. The first problem is harder. If for a given M the construction of M-closure is unlimited then the using of M-deductive basis is unprofitable.

Example 2. Given $U = \{1,2,3,4\}$, $G = \mathbb{N}'$,

$\mathcal{B}_1 = P(x,y,z,u') \& P(x,y,z',u) \longrightarrow P(x,y,z,u)$,

$\mathcal{B}_2 = P(x',y,z,u) \& P(x,y',z,u) \longrightarrow P(x,y,z,u)$,

$M = \{\mathcal{B}_1, \mathcal{B}_2\}$,

$r_1 = (0,0,0,0)$ and for $i \geq 1$

$r_{2i}(\{1,2,3\}) = r_{2i-1}(\{1,2,3\})$, $r_{2i}(4) = r_{2i-1}(4) + 1$,

$r^+_{2i}(\{1,2,4\}) = r_{2i-1}(\{1,2,4\})$, $r^+_{2i}(3) = r_{2i-1}(3) + 1$,

$r_{2i+1}(\{1,3,4\}) = r_{2i}(\{1,3,4\})$, $r_{2i+1}(2) = r_{2i}(2) + 1$,

$r^+_{2i+1}(\{2,3,4\}) = r_{2i}(\{2,3,4\})$, $r^+_{2i+1}(1) = r_{2i}(1) + 1$.

Let be $R_1 = \{r_1\}$ and for $i \geq 2$

$R_i = R_{i-1} \setminus \{r_{i-1}\} \cup \{r_i, r_i^+\}$.

Then holds $(0,0,0,0) \in /R_{i+1}/_{M,i}$ and

$(0,0,0,0) \notin /R_{i+1}/_{M,i-1}$,

i.e. $/R_{i+1}/_{M,i} \neq /R_{i+1}/_{M,i-1}$.

We get that for all natural numbers i exists a relation R with $/R/_M \neq /R/_{M,i}$.

2. A solution of the first problem

Example 2 shows that there are sets of TGD and of DD without bounds for closure.

A set of DD M is called Sheffer-set if there is a DD $\mathcal{B}$ with $M \subseteq \{\mathcal{B}' / \{\mathcal{B}\} \models \mathcal{B}'\}$, $M \models \mathcal{B}$.

Theorem 1. Given a Sheffer-set M of DD. For any relation R holds $/R/_M = /R/_{M,1}$.

Theorem 1 cannot be extended to TGD because the following TGD

$$P(x,y',z,u') \,\&\, P(x',y,z,u') \,\&\, P(x,y',z',u) \,\&$$
$$\&\, P(x',y,z',u) \longrightarrow P(x,y,z,u)$$

is equivalent to the set M in example 2.

Proof.

For the proof of theorem 1 we use the approach of /3/ to recursive axioms. We introduce some useful definitions. Given a TGD β with the set $Var(\beta)$ of variables and a subset V of $Var(\beta)$, a substitution $S_{\tilde{x}}^{\tilde{y}}$ of old variables $\tilde{x} = (x_1,\dots,x_k)$ and corresponding new variables $\tilde{y} = (y_1,\dots,y_k)$. The substitution $S_{\tilde{x}}^{\tilde{y}}$ is said to be safe for β and V if holds $\{y_1,\dots,y_k\} \cap Var(\beta) = \emptyset$ and $\{x_1,\dots,x_k\} \cap V = \emptyset$.

Given two formulaes $\beta = \beta_1 \& \dots \& \beta_p$ and $\gamma = \gamma_1 \& \dots \& \gamma_q$ with atomar formulaes $\beta_1,\dots,\beta_p$, $\gamma_1,\dots,\gamma_q$. Given $V \subseteq Var(\beta) \cup Var(\gamma)$.

A pair of safe substitutions $<S_1,S_2>$ for β resp. γ and V is called safe unificator if holds $\{S_1(\beta_1),\dots,S_1(\beta_p)\} \subseteq \{S_2(\gamma_1),\dots,S_2(\gamma_q)$.

Now we consider special derivations $\Delta(M,P(x))$ for formulaes $P(\tilde{x})$:

$\beta_1,\beta_2,\dots,\beta_i,\dots$ with $\beta_1 = P(\tilde{x})$ and for any i, $\beta_{i-1} = \beta_{i-1}^1 \& \dots \& \beta_{i-1}^k$, $\beta_i = \beta_i^1 \& \dots \& \beta_i^{k'}$ there exist some j, $P(\tilde{y}_1) \& \dots \& P(\tilde{y}_s) \longrightarrow P(\tilde{y}) \in M$ and a safe substitution S with

$\beta_i^m = \beta_{i-1}^m \quad (1 \le m \le j)$, $\beta_i^{j+s+l} = \beta_{i-1}^{j+l} \quad (1 \le l \le k-j+1)$,
$\beta_{i-1}^{j+1} = S(P(\tilde{y}))$, $\beta_i^{j+t} = S(P(\tilde{y}_t)) \quad (1 \le t \le s)$

and $k' = k+s-1$.

Any such derivation $\Delta(M,P(x))$ correspond to the generation of a new element in $/R/_{M,i}$ for a interpretation I and vice versa.

Therefore our first problem is solved if a halting condition exists for derivations.

Corollary 2 (/1,3/). Given some derivation $\Delta(M,P(x)) = \beta_1,\dots,\beta_i,\dots$ If for some j, a safe unificator exists for $<\beta_j,\beta_{j+1}>$ then the derivation $\Delta(M,P(x))$ is equivalent to the derivation $\beta_1,\dots,\beta_j$.

Example 3 (Continuation of example 2).

$$\Delta(\{\beta_1,\beta_2\},P(x,y,z,u)) =$$

$$P(x,y,z,u) \;,$$

$$P(x,y,z,u') \;\&\; P(x,y,z',u) \;,$$

$$P(x,y',z,u') \;\&\; P(x',y,z,u') \;\&\; P(x,y,z',u) \;,$$

$$P(x,y',z,u'') \;\&\; P(x,y',z'',u') \;\&\; P(x,y,z',u) \;\& \\ \& \; P(x',y,z,u') \;,$$

$$P(x,y'',z,u'') \;\&\; P(x',y',z,u'') \;\&\; P(x,y',z'',u') \;\& \\ \& \; P(x',y,z,u') \;\&\; P(x,y,z',u) \;, \dots$$

Obviously, for no j a safe unification exists for $<\beta_j,\beta_{j+1}>$ and $V = \{x,y,z,u\}$.

Now we can prove theorem 1.

Given a DD β . Any derivation $\Delta(\{\beta\},P(\tilde{x})) = \beta_1,\dots,\beta_i,\dots$ is equivalent to $\{\beta_1,\beta_2\}$.

Let be $\beta = P(\tilde{x}_1) \;\&\; \dots \;\&\; P(\tilde{x}_k) \longrightarrow P(\tilde{x}_o)$,

$\beta_1 = P(\tilde{x})$, $\beta_2 = \beta_{21} \;\&\dots\&\; \beta_{2k}$,

$\beta_3 = \beta_{31} \& \ldots \& \beta_{3\,(2k-1)}$, ...

By definition of β_i we get for some i , j :

$\beta_{2l} = \beta_{3l}$ $(1 \leq l \leq j)$, $\beta_{3\ j+k+i} = \beta_{2\ j+i}$ $(1 \leq i \leq k-j)$.

A safe unification exists for the pair

$< \beta_{2\ j+1} , \beta_{3\ j+1} \& \ldots \& \beta_{3\ j+k} >$ and therefore also for $< \beta_2 , \beta_3 >$.

Given a Sheffer-set M of DD with $M = \{ \beta' \ / \ \{\beta\} \models \beta' \}$. It is easy to prove that for every derivation $\Delta(M,P(\tilde{X}))$ exists an equivalent derivation $\Delta'(\{\beta\},P(\tilde{X}))$.

3. Characterizations of Sheffer-sets

In /2/ a characterization of Sheffer-sets is given in the special case of binary JD. This result can be extendee to bigger classes of JD.

A set of JD K is called $\cap$-closed if holds $(X_1,\ldots,X_{i-1},X_i \cap Y,X_{i+1},\ldots,X_k) \in K$ for any $(X_1,\ldots,X_k)$, $(X_1,X_2,\ldots,X_{i-1},Y,X_{i+1},\ldots,X_k) \in K$.

Corollary 3. Given a DD β and the corresponding JD D . The set $K = \{ D' \ / \ \{D\} \models D' \}$ is $\cap$-closed.

A JD (DD) $(X_1,\ldots,X_m)$ is called full first-order hierarchical dependency (FOHD) if holds $X_i \cap X_j = X_k \cap X_l$ for any pairs with $i \neq j$, $k \neq l$.

In /4/ is preved

Theorem 2. Let K be a set of FOHD. The following are equivalent:

(1) K is equivalent to a single JD D .

(2) K is $\cap$ -closed.

The characterization problem for Sheffer-sets is open for arbitrary sets of JD.

Given two JD's $D = (X_1,\ldots,X_m)$, $D' = (Y_1,\ldots,Y_l)$.
If for any X_i there is a Y_j such that $X_i \subseteq Y_j$ it holds $D \leq D'$. If for any Y_j there is some X_i such that $X_i = Y_j$ it holds $D \subseteq D'$.

A set K is called $\subseteq$-closed

(i) if for any JD D' and $D = (X_1,\ldots,X_m) \in K$ with $D \models D'$ $D' \in K$ and

(ii) if for $D = (X_1,\ldots,X_m)$, $D' = (Y_1,\ldots,Y_l) \in K$ there exist i , j $(2 \leq i \leq m,\ 1 \leq j \leq l-1)$ such that for $D'' = (Y_1,\ldots,Y_j,X_i,\ldots,X_m)$ holds $D \leq D''$, $D \subseteq D''$, $D' \leq D''$, $D' \subseteq D''$.

Corollary 4. $\subseteq$-closed sets are Sheffer-sets.

References

/1/ C.L. Chang, R.C.-T. Lee, Symbolic logic and mechanical theorem proving. New York 1973.

/2/ N. Goodman, Y.C. Tay, A characterization of multivalued dependencies equivalent to a join dependancy. IPL 18(1984), 261 - 266.

/3/ J. Minker, J.M. Nicolas, On recursive axioms in deductive databases. IS 8, 1, 1-13, 1983.

/4/ B. Thalheim, Abhängigkeiten in Relationen. Dissertation, Dresden 1985.

/5/ B. Thalheim, Bibliographie zur Theorie der Abhängigkeiten in Relationen 1970 - 1984. Dresden 1985.

ON THE CONNECTION BETWEEN MINIMAL KEYS AND RELATIONS

VU DUC THI

Computer and Automation Institute,Hungarian Academy of Sciences

Budapest,Hungary

§0.INTRDDUCTION

The relational datamodel which was defined by E.F.Codd [1] is one of the possible datamodels of a data base.In this model the form of data storage is a matrix /relation/.

The minimal keys play important roles for the logic and structural investigation of this model.The equivalence of minimal keys with Sperner-systems was proved by J.Demetrovics [2] .

The main purpose of this paper is to give two effective combinational algorithms.The first algorithm,for a given Sperner-system K,determines a relation R so that the set of minimal keys of R is exactly K. Conversely, for a given arbitrary relation R the second algorithm determines the set of minimal keys of R.

§1.DEFINITIONS

We start with some necessary definitions.

Definitionl.l. Let $R=\{h_1,\dots,h_m\}$ be a relation over Ω,and $A,B \subseteq \Omega$. Then we say that B functionally depends on A in R if

$$(\forall h_i,h_j \in R)\Big((\forall a \in A)(h_i(a) = h_j(a)) \longrightarrow (\forall b \in B)(h_i(b)= h_j(b))\Big),$$

where $1 \leq i \leq m, 1 \leq j \leq m$.

Denote this by $A \xrightarrow{R} B$.

Definition 1.2. Let R be a relation overΩ,and $A \subseteq \Omega$ .Then A is a minimal key of R if $A \xrightarrow{R} \Omega$ and for any $B \subseteq A$, $(B \xrightarrow{R} \Omega) \longrightarrow (B=A)$.

Let us denote by K_R the set of all minimal keys of R.It is clear that K form Sperner-system.

For a Sperner-system K we can define the set of antikeys , denoted by K ,as follows:

$K^{-1} = \{ A \subsetneqq \Omega : (B \in K) \to (B \nsubseteq A)$ and $(A \subsetneqq C) \to (\exists B \in K)(B \subseteq C) \}$.

It is easy to see that K^{-1} is also a Sperner-system, K and K^{-1} are uniquely determined by each other .

In this paper we assume that Sperner-systems playing the role of the set of minimal keys /antikeys/ are not empty / don't contain the full set Ω /

Definition 1.3. Let K be a Sperner-system over Ω . We say that a relation R represents K if $K_R = K$.

§2. THE FIRST ALGORITHM

The first time, we construct an algorithm which determines the set of antikeys from a given Sperner-system.

Let us given an arbitrary Sperner-system $K = \{B_1 ,\ldots, B_m\}$ over Ω .We set $K_1 = \{\Omega \setminus \{a\} : a \in B_1\}$ .It is obvious that $K_1 = \{B_1\}^{-1}$.

Let us suppose that we have constructed $K_q = \{B_1, \ldots, B_q\}^{-1}$ for $q < m$ .We assume that $X_1, \ldots, X_{t_q}$ are the elements of K_q containing B_{q+1} .

So $K_q = F_q \cup \{X_1, \ldots, X_{t_q}\}$, where $F_q = \{A \in K_q : B_{q+1} \nsubseteq A\}$.

For all i $(i=1,\ldots,t_q)$, we construct the antikeys of $\{B_{q+1}\}$ on X_i in the analogous way as K_1 , which are the maximal subsets of X_i not containing B_{q+1} . We denote them by $A_1^i ,\ldots, A_{r_i}^i$ $(i=1,\ldots,t_q)$.

Let $K_{q+1} = F_q \cup \{A_p^i : A \in F_q \to A_p^i \not\subset A , 1 \leq i \leq t_q , 1 \leq p \leq r_i\}$.

Theorem 2.1. ([4]) For every q $(1 \leq q \leq m)$, $K_q = \{B_1 ,\ldots, B_q\}^{-1}$,i.e. $K_m = K^{-1}$.

Because K and K^{-1} are uniquely determined by each other , the determination of K^{-1} based on our algorithm does not depend on the order of $B_1 ,\ldots, B_m$.

Now we assume that the elementary step being counted is the comparison of two attribute names. Consequently, if we assume that subsets of Ω are represented as sorted lists of attribute names ,then a Boolean operation on two subsets of Ω requires at most $|\Omega|$ elementary steps.

According to the construction of K_q . We have

$K_q = F_q \cup \{X_1, \ldots, X_{t_q}\}$, where $1 \leq q \leq m-1$.

Denote ℓ_q the number of elements of K_q .It is clear that for constructing K_{q+1} the worst- case time of algorithm is $O(n^2(\ell_q - t_q)t_q)$ if $t_q < \ell_q$ and $O(n^2 t_q)$ if $\ell_q = t_q$. Consequently, the total time spent by the algorithm in the worst-case is $O(n^2 \sum_{q=1}^{m-1} t_q u_q)$, where $|\Omega| = n$,

$$u_q = \begin{cases} \ell_q - t_q & \text{if } \ell_q > t_q , \\ 1 & \text{if } \ell_q = t_q . \end{cases}$$

It can be seen that when there are only a few minimal keys / that is m is small /our algorithm is very effective . In cases for which $\ell_q \leq \ell_m (\forall q : 1 \leq q \leq m-1)$ it is clear that our algorithm requires a number of elementary operations which is not greater than $O(n^2 |K| . |K^{-1}|^2)$. Thus, in these cases our algorithm finds K^{-1} in polynomial time in $|\Omega|$, $|K|$ and $|K^{-1}|$.

In [8] , we have proved that the worst-case time of our algorithm is exponential in the number of attributes.

Remark 2.2. Let $K^{-1} = \{A_1, \ldots, A_t\}$ be a set of antikeys.

Let $R = \{h_0, h_1, \ldots, h_t\}$ be a relation over Ω given as follows :

For all $a \in \Omega$, $h_0(a) = 0$,

$$\text{For } i \; (1 \leq i \leq t), \; h_i(a) = \begin{cases} 0 & \text{if } a \in A_i , \\ i & \text{otherwise.} \end{cases}$$

In [3] , it has been proved that R represents K.

Thus, for a given Sperner-system K we can construct an algorithm which determines a relation R so that the set of minimal keys of R is exactly K ,as follows:

Algorithm 2.3.

Step 1 : based on Algorithm 2.1 we construct K^{-1} .

Step 2 : by Remark 2.2 we have R which represents K .

It is clear that the complexity of this algorithm is the complexity of Algorithm 2.1 .

§3. THE SECOND ALGORITHM

In [3] , the equality sets of the relation are defined as follows: Let $R = \{h_1, \dots, h_m\}$ be a relation over Ω .We denote by E_{ij} the set $\{a \in \Omega : h_i(a) = h_j(a)\}$,where $i \neq j$, $1 \leq i \leq m$ and $1 \leq j \leq m$.

We set $M = \{E_{ij} : \overline{\exists} \; E_{qp}$ such that $E_{ij} \subset E_{qp}\}$.

Practically, it is possible that there are some E_{ij} which are equal to each other . We choose one such E_{ij} to M , .i.e. the elements of M are not equal to each other.

Theorem 3.1. ([4]) $K_R^{-1} = M$, i.e. M is the set of antikeys of R .

It can be seen that the complexity of finding the set of antikeys of R is polynomial in $|\Omega|$ and $|R|$,where $|\Omega| = n$, $|R| = m$.

Let $G \subseteq P(\Omega)$, $\emptyset \notin G$ and $A \subseteq \Omega, A \neq \emptyset$ .We define MIN(G) and V(A) , as follows :

MIN(G)= B ,where B is an element of G such that $|B| = \min\{|B_i| : B_i \in G\}$.

V(A)= a , where a is an element of A .

Let K be a Sperner-system over Ω , $K^{-1} = \{B_1, \dots, B_m\}$,and let B be a key ,i.e. there is an $A \in K$ so that $A \subseteq B$.

We set $B_i^1 = B \cap \overline{B}_i (\forall i : 1 \leq i \leq m)$, $r_1 = m$ and $G_1 = \{B_1^1, \dots, B_{r_1}^1\}$.

Then for i $(i \geq 1)$ let

$$F_{i+1} = \{B_j^i \in G_i : V(\mathrm{MIN}(G_i)) \notin B_j^i\} \text{ and}$$

$$G_{i+1} = \{B_j^i - \mathrm{MIN}(G_i) : B_j^i \in F_{i+1}\} = \{B_1^{i+1}, \dots, B_{r_{i+1}}^{i+1}\} .$$

Clearly, because $|\Omega|$ is finite there exists a natural number p such that $G_p \neq \emptyset$ and $F_{p+1} = \emptyset$. It is obvious that $p \leq \min(n,m)$, where $|\Omega| = n$.

Theorem 3.2. ([6]) $A = \bigcup_{i=1}^{p} \{V(MIN(G_i))\}$ is an element of K. Thus, A is a minimal key.

It is easy to see that the worst-case time of finding the element A is $O(n^2m)$.

Theorem 3.3. ([7]) Let H be a Sperner-system over Ω, and let $H^{-1} = \{B_1,\ldots,B_m\}$ be a set of antikeys of H, $T \subseteq H$. Then $T \subsetneq H$ and $T \neq \emptyset$ if and only if there is a $B \subseteq \Omega$ such that $B \in T^{-1}$ and $B \not\subseteq B_i (\forall i: 1 \leq i \leq m)$.

Let K be a Sperner-system over Ω : Based on Theorem 3.3 we can construct a following algorithm which determines a set H so that $H^{-1} = K$ by induction.

Algorithm 3.4.

Step 1: by Algorithm 3.2 we construct a minimal key A_1. Set $K_1 = \{A_1\}$.

Step i+1 : if there exists a $B \in K_i^{-1}$ such that $B \not\subseteq B_j (\forall j : 1 \leq j \leq m)$. then by Algorithm 3.2 we determines a minimal key $A_{i+1} (A_{i+1} \subseteq B)$. After that, let $K_{i+1} = K_i \cup A_{i+1}$. In the converse case we set $H = K_i$.

It is easy to see that there is a natural number p so that $K_p = H$.

It can be seen that Algorithm 3.4 is very effective if the number of minimal keys is small.

In [7], we have been shown that in many cases the complexity of Algorithm 3.4 is polynomial in $|\Omega|, |K|$ and $|H|$, the worst-case time of this algorithm is exponential in the number of attributes.

For a given arbitrary relation R we construct a following algorithm which determines the set of minimal keys of R.

Algorithm 3.5.

Step 1 : according to Theorem 3.1 we construct the set of antikeys of R.

Step 2 : based on Algorithm 3.4 we have the set of minimal keys of R.
It is clear that the worst-case time of Algorithm 3.5 is
max $\{$ the complexity of Algorithm 3.4,the complexity of Algorithm3.
It is easy to see that we can use Algorithm 2.1 and 3.4 for determining special Sperner-systems,i.e. sets of special minimal keys and antikeys (see [5]) .

ACKNOWLEDGEMENTS

The author would like to take this opportunity to express deep gratitude to Professor J.Demetrovics for his help,valuable comments and suggestions.

REFERENCES

[1] Codd E.F. , Relational model of data for large shared data banks. Communications of the ACM,13, (197o) 377-384.

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

[3] Demetrovics J.,Relációs adatmodell logikai és strukturális vizsgálata.MTA -SZTAKI Tanulmányok ,Budapest ,114 (198o) 1-97.

[4] Vu duc Thi ,Remarks on closure operations.
MTA-SZTAKI Közlemények ,Budapest , 3o (1984) 74-87.

[5] Vu duc Thi ,Some special Sperner-systems.
MTA-SZTAKI Közlemények ,Budapest,32 (1985) 163-174.

[6] Vu duc Thi ,Algorithms for finding minimal keys and antikeys of relational data base.MTA-SZTAKI Közlemények,Budapest,33 (1985) 113-143.

[7] Vu duc Thi ,Relációs adatmodell antikulcsairól.
Alkalmazott matematikai Lapok (to appear) .

[8] Vu duc Thi , Minimal keys and antikeys.
Acta Cybernetica (to appear)

SOME ADDITIONAL PROPERTIES OF KEYS FOR RELATION SCHEME

Ho Thuan

MTA - SZTAKI

Abstract.

In this paper we prove some additional properties of keys and superkeys for relation schemes . Some of them and their variants have been proved / perhaps by different methods / and used to design an algorithm to find all keys for any relation scheme / 3 / .

§ I. Introduction.

In [I] some characteristic properties of keys for a given relation scheme $S = \langle \Omega , F \rangle$ have been investigated , in particular the necessary condition under which a subset X of Ω is a key for S .

In this paper , we prove some additional properties of keys and superkeys for relation schemes. In particular , sufficient conditions under which a superkey in a special family is a key for a given relation scheme are established.

Finally, some remarks for improving the performance of the algorithm of Lucchesi and Osborn [3] are also given.

The notation used here is the same as in [I] and [2] .

The reader is required to know the basic notation of the relational model and functional dependency [4] .

§ 2.

In this section we recall some notions and results which will be needed in the sequel.

Let $S = \langle \Omega , F \rangle$ be a relation scheme, where

$$\Omega = \{ A_I , A_2 , \dots , A_n \}$$

$$F = \{ L_i \longrightarrow R_i \mid i = I, 2, \dots , k \, ; \, L_i, R_i \subseteq \Omega \}$$

Let us denote :

$$L = \bigcup_{i=1}^{k} L_i \quad , \quad R = \bigcup_{i=1}^{k} R_i$$

$$\mathcal{K}_S = \{ K \mid K \text{ is a key for } S \} \; ,$$

$$C_i = \Omega \setminus L_i^+ \quad , \quad i = I, 2, \dots , k \, ;$$

$$\mathcal{J} = \{ i \mid \text{there is no j such that } L_i \supset L_j \}$$

$$\subseteq \{ I, 2, \dots , k \} .$$

Recall that for $X \subset \Omega$,

$$X^+ = \{ A \mid (X \longrightarrow A) \in F^+ \}$$

is the closure of X w.r.t. F , where F^+ is the closure of F , i.e. the set of all FDs that can be inferred from the FDs in F by repeated application of Armstrong's axioms [5] .

Two subsets X and Y of Ω are said equivalent, written $X \longleftrightarrow Y$, if $X \xrightarrow{*} Y$ and $Y \xrightarrow{*} X$.+)

It is easy to show that

+) In the following $X \xrightarrow{*} Y$, XY stand for $(X \rightarrow Y) \in F^+$ and $X \cup Y$ respectively.

$$X \leftrightarrow Y \quad \text{iff} \quad X^+ = Y^+ .$$

Without loss of generality , throughout this paper we assume that

$$L_i \cap R_i = \emptyset \ , \ i = I, 2, \dots , k \ ;$$

$$L_i \neq L_j \quad \text{with} \quad i \neq j \ .$$

We have the following lemmas :

Lemma I. [2] .

Let $S = \langle \Omega , F \rangle$ be a relation scheme, $X , Y \subseteq \Omega$, then $(X Y)^+ = (X^+ Y)^+ = (X Y^+)^+$

Lemma 2. [2] .

For any $i \in \mathcal{J}$,

L_i is a key for S if and only if $C_i = \emptyset$.

Lemma 3. [2] . (Key representation) .

Let $S = \langle \Omega , F \rangle$ be a relation scheme. Then any key K for S has the following form

$$K = L_i X_i$$

for some $i \in \mathcal{J}$, where $X_i \subseteq C_i$.

Remark I.

It is easy to see that for every $j \in \{ I, 2, \dots, k \}$, $L_j C_j$ is a superkey for S; and for every $X \subseteq \Omega$, X is a superkey for S iff $X^+ = \Omega$.

§ 3.

We are now in a position to prove some additional properties of keys and superkeys for relation schemes which can be used for the design of algorithms to find keys for relation scheme.

Lemma 4.

Let $S = \langle \Omega , F \rangle$ be a relation scheme.

Then $\forall \ i \neq j \ , i, j \in \{ I, 2, \ldots, k \}$

$L_i \left(C_i \cap L_j C_j \right)$ is a superkey for S .

Proof.

The case $C_i = \emptyset$, we have

$$L_i \left(C_i \cap L_j C_j \right) = L_i \quad .$$

But in that case, it is obvious that L_i is a superkey.

Now we consider the case $C_i \neq \emptyset$.

First , we will prove that if $C_i \neq \emptyset$ then

$$C_i \cap L_j C_j \neq \emptyset \ , \quad \forall \ j \neq i \ .$$

In fact, assume the contrary that

$$C_i \cap L_j C_j = \emptyset .$$

It follows that :

$$(C_i \cap L_j)\ (C_i \cap C_j) = \emptyset .$$

On the other hand,

$$C_i = (C_i \cap L_j)\ (C_i \cap C_j)\ (C_i \cap (L_j^+ \setminus L_j)) = C_i \cap (L_j^+ \setminus L_j) ,$$

showing that

$$C_i \subseteq (L_j^+ \setminus L_j) .$$

Thus

$$\Omega \setminus C_i \supseteq \Omega \setminus (L_j^+ \setminus L_j)$$

or $$L_i^+ \supseteq L_j\ C_j .$$

The last set inclusion shows that L_i is a superkey, a contradiction.

Therefore, if $C_i \neq \emptyset$ then $C_i \cap L_j\ C_j \neq \emptyset$.

Now, it is clear that

$$L_i \xrightarrow{*} L_i^+$$

$$C_i \cap L_jC_j \xrightarrow{*} C_i \cap L_jC_j .$$

Consequently,

$$L_i\ (C_i \cap L_j\ C_j) \xrightarrow{*} L_i^+\ (C_i \cap L_j)\ (C_i \cap C_j)$$

On the other hand , we have :

$$L_j = (L_j \setminus C_i)\ (L_j \cap C_i) \subseteq L_i^+\ (C_i \cap L_j) .$$

$$C_j = (C_j \setminus C_i)\,(C_j \cap C_i) \subseteq L_i^+ \,(C_i \cap C_j) .$$

Hence

$$L_i^+ \,(C_i \cap L_j)(C_i \cap C_j) \supseteq L_j \; C_j \quad .$$

Finally, we have

$$L_i \,(C_i \cap L_j \; C_j) \xrightarrow{*} L_j \; C_j$$

showing that $L_i \,(C_i \cap L_j \; C_j)$ is a superkey for S .

Lemma 5.

Let K be any key for $S = \langle \Omega , F \rangle$ and having the form

$$K = L_i \; X \quad , \quad X \subset C_i .$$

Then there exists $j_o \neq i$ such that

$$K \subseteq L_i \,(C_i \cap L_{j_o} \; C_{j_o}) .$$

Proof.

Assume the contrary that

$$L_i \; X \not\subseteq L_i \,(C_i \cap L_j \; C_j) \quad , \forall\, j \neq i ,$$

or, equivalently

$$X \not\subseteq C_i \cap L_j \; C_j \quad , \forall\; j \neq i \quad .$$

Then, for all $j \neq i$,there exists an attribute

$$A_{i_j} \in (L_j^+ \setminus L_j) \cap X \quad .$$

Obviously, we have :

$$L_i \quad X \xrightarrow{*} L_i \quad R_i \quad X \quad .$$

Then there must exist p such that $L_p \subseteq L_i \; R_i \; X$.

(Otherwise, $L_i \; X \not\xrightarrow{*} \Omega$, a contradiction).

Let $A_{i_p} \in (L_p^+ \setminus L_p) \cap X$ and let $X' = X \setminus \{ A_{i_p} \}$

Since $A_{i_p} \notin L_p$, so $L_p \subseteq L_i \; R_i \; X'$

Therefore , it is easy to see that

$$L_i \; X' \xrightarrow{*} L_i \; R_i \; X' \xrightarrow{*} L_i \; R_i \; L_p \; R_p \; X'$$

$$\xrightarrow{*} L_i \; R_i \; L_p^+ \; X' \quad .$$

Moreover $A_{i_p} \in L_p^+$.

Consequently,

$$L_i \; X' \xrightarrow{*} L_i \; X \qquad ,$$

showing that L_i X is not a key, a contradiction .

The proof is complete .

<u>Corollary I</u>.

The family

$$\{ L_i (C_i \cap L_j \; C_j) \mid j \neq i \; , \; I \leq i, j \leq k \}$$

can be used to find all keys for the relation scheme S.

<u>Remark 2</u>.

Lemmas 4 and 5 have been proved / perhaps by different methods / and used to design an interesting algorithm to find all keys for any relation scheme [6] .

<u>Theorem I</u>.

Let $S = \langle \Omega , F \rangle$ be a relation scheme. Suppose that the following conditions hold :

i/ $L_i (C_i \cap L_j \; C_j) = L_i \; C_i$, $\forall$ i = I, 2,..., k;

ii/ $L_i \cap R_j = \emptyset$, $\forall$ j $\neq$ i .

Then $L_i \; C_i$ is a key for S .

<u>Proof</u>.

First, from condition i/ , we can prove that for every $\bar{X} \subset C_i$, $L_i \; \bar{X}$ is not a superkey for S .

In fact, since $C_i \cap L_j \; C_j = C_i$, $\forall$ j $\neq$ i , it follows that

$$C_i \cap (L_j^+ \setminus L_j) = \emptyset \; , \; \forall \; j \; .$$

Therefore, if $A \in C_i$ then

$$\{ A \} \cap (L_j^+ \setminus L_j) = \emptyset \; , \; \forall \; j$$

Let A be any element of C_i and $X = C_i \setminus \{ A \}$.

It is easy to see that

$$L_i \; X \xrightarrow{*} L_i \; R_i \; X$$

Since $L_i \; R_i \cap C_i = \emptyset$ / because $L_i \; R_i \subseteq L_i^+$ /, $A \in C_i$, $A \notin X$, it follows that

$$A \notin L_i \; R_i \; X \quad .$$

Now, suppose that there exists

$$L_h \subseteq L_i \; R_i \; X \quad , \quad h \neq i$$

Obviously $A \notin L_h$ and

$$L_i \; X \xrightarrow{*} L_i \; R_i \; X \xrightarrow{*} L_i \; R_i \; L_h \; R_h \; X \; .$$

It is clear that $A \notin R_h$, otherwise

$$A \in \left(L_h^+ \setminus L_h \right) \text{ , a contradiction .}$$

By repeating the same reasoning, we can prove that

$$L_i \; X \quad \overset{*}{\not\longrightarrow} \quad \Omega \; ,$$

showing that for every $\bar{X} \subset C_i$, $L_i \; \bar{X}$ is not a superkey for S.

In other words $L_i \; C_i$ contains only key / or keys / of the form $L_i' \; C_i$ with $L_i' \subseteq L_i$.

By condition ii/ , we have

$$L_i = L_i \setminus R \subseteq L \setminus R$$

On the other hand, from [I]

$$L \setminus R \subseteq \Omega \setminus R \subseteq K \quad , \forall K \in \mathcal{K}_S \; .$$

This shows that $L_i \; C_i$ is a key for S . Q.E.D.

Corollary 2.

If $S = \langle \Omega , F \rangle$ has a key $K = L_i \; X$ with $X \subset C_i$ then there exists $j_o \neq i$ such that

$$L_i \; \left(C_i \cap L_{j_o} \; C_{j_o} \right) \subset L_i \; C_i$$

Corollary 3.

If $L_i \left(C_i \cap L_j \; C_j \right) = L_i \; C_i$, $\forall j \neq i$, then $C_i \subseteq H = \bigcup_{K \in \mathcal{H}_S} K$

In otherwords, C_i consists of only prime attributes.

Corollary 4.

If $|C_i| = I$, $\forall\ i = I, 2, \dots, k$, then $L_j\ C_j$ is a key for S iff there is no q, $q \neq j$, such that

$$L_j\ C_j \supset L_q\ C_q$$

Theorem 2.

Let $S = \langle \Omega, F \rangle$ be a relation scheme, $L_i\ Z$ be a key for S,

$$L_i \longleftrightarrow L_j \quad , \quad L_i \cap Z = L_j \cap Z = \emptyset \quad ;$$

$$L_j \cap R_h = \emptyset, \ \forall\ h \neq j$$

Then $L_j\ Z$ is a key for S .

Proof.

It is easy to see that if $L_i\ Z$ is a key for S and $L_i \leftrightarrow L_j$ then $L_j\ Z$ is a superkey for S.

In fact we have

$$L_j\ Z \xrightarrow{*} L_i\ Z \xrightarrow{*} \Omega$$

Moreover, we can prove that for every $\bar{Z} \subset Z$, $L_j\ \bar{Z}$ is not a superkey for S. Assume the contrary that $L_j\ \bar{Z}$ is a superkey for S with $\bar{Z} \subset Z$.

It is clear that

$$\Omega = (L_j\ \bar{Z})^+ = (L_j^+\ \bar{Z})^+ = (L_i^+\ \bar{Z})^+ = (L_i\ \bar{Z})^+$$

showing that L_iZ is not a key , a contradiction .

The condition $L_j \cap R_h = \emptyset$, $\forall$ h

implies that $L_j \cap R = \emptyset$.

Hence $L_j \subseteq L \setminus R$.

Moreover, again by [I]

$$L \setminus R \subseteq \Omega \setminus R \subsetneq K \ , \ \forall K \in \mathcal{K}_S$$

showing that L_jZ is a key for S .

Theorem 3.

Let $S = \langle \Omega , F \rangle$ be a relation scheme ,

$X, Y, Z \subseteq \Omega$, $X \cap Z = Y \cap Z = \emptyset$.

Suppose that the following conditions hold :

i/ $X \longleftrightarrow Y$;

ii/ For every $X' \subset X$ with $|X'| = |X| - I$ there exists $Y' \subset Y$ such that $Y' \longleftrightarrow X'$,

iii/ For every $Y' \subset Y$ with $|Y'| = |Y| - I$ there exists $X' \subset X$ such that $X' \longleftrightarrow Y'$.

Then ZX is a key iff ZY is a key .

Proof.

ONLY IF. Suppose that ZX is a key .

Since $X \longleftrightarrow Y$, following the proof of theorem 2, YZ is a superkey for S while YZ' is not for every $Z' \subset Z$.

In other words, YZ contains only key / or keys / of the form $Y'Z$ with $Y' \subseteq Y$.

Now, we shall prove that for every $\bar{Y} \subset Y$, $\bar{Y} \ Z$ is not a superkey for S.

The proof is by contradiction. Let $Y'Z$ is a superkey for S with $\bar{Y} \subseteq Y' \subsetneq Y$ where $|Y'| = |Y| - 1$.

Taking the condition iii/ into account, we have

$$\Omega = (Y'Z)^+ = ((Y')^+ Z)^+ = ((X')^+ Z)^+ = (X' Z)^+$$

Where $X' \subset X$, $X' \longleftrightarrow Y'$,

showing that XZ is not a key , a contradiction .

Similarly we can prove the IF part .

The proof is complete .

Corollary 5.

Let $S = \langle \Omega , F \rangle$ be a relation scheme,

$L_i \longleftrightarrow L_j$, $|L_i| = |L_j| = I$, $L_i \cap Z = L_j \cap Z = \emptyset$.

Then $L_i\ Z$ is a key for S iff $L_j\ Z$ is a key .

Proof.

It is easy to verify that all conditions of theorem 3 are satisfied .

Example I.

We take up again the example in [3] . According to our notation, we have

$$\Omega = \{ C, I, N, P, T \}^{+/}$$

$$F = \{ N \longrightarrow I,\ I \longrightarrow N,\ NC \longrightarrow PT,\ PT \longrightarrow C \}$$

It is obvious that $N \longleftrightarrow I$.

So, using the algorithm Lucchesi and Osborn, after the keys IPT and IC have been found, we can add immediately to the set $\mathbb{K}$ two new keys NPT and NC.

Theorem 4.

Let $S = \langle \Omega , F \rangle$ be a relation scheme, $L_i\ Z$ is a key for S with $Z \cap L_i = \emptyset$.

If $Z \subset C_j$, $L_j \longleftrightarrow L_i$

and $L_j\ (C_j \cap L_h\ C_h) = L_j\ C_j$, $\forall h \neq j$

then S has no key including L_j .

Proof.

The condition $L_j \longleftrightarrow L_i$ implies that $L_j\ Z$ is a superkey for S .From $Z \subset C_j$, it follows that $L_j\ C_j$ is not a key. From $L_j\ (C_j \cap L_h\ C_h) = L_j\ C_j$ and $L_j\ C_j$ is not a key, by corollary 2, we conclude that S has no key that includes L_j . Q .E.D.

+/ C, I, N, P, T stand for Course, ID- number, Name, Professor and Time respectively.

§ 4.

In [3] C.L. Lucchesi and S.L. Osborn provided a very intteresting algorithm to determine the set of all keys for any relation scheme $S = \langle \Omega , F \rangle$.
The algorithm has time complexity

$$O\left(|F|\,|\mathcal{K}_S|\,|\Omega|\,(|\mathcal{K}_S| + |\Omega|)\right) ,$$

in our notation, i.e. in time polynomial in $|\Omega|$, $|F|$ and $|\mathcal{K}_S|$.
We copy here this algorithm with some modifications in accordance to our notation .

<u>ALGORITHM OL1</u>. Set of all keys for $S = \langle \Omega , F \rangle$;
<u>Comment</u> $\mathcal{K}_S$ is the set of keys being accumulated in a sequence which can be scanned in the order in which the keys are entered;

```
$\mathcal{K}_S \leftarrow \{ \mathrm{Key}(\Omega , F , \Omega) \}$[+/];
for each K in $\mathcal{K}_S$ do
    for each FD $(L_i \rightarrow R_i)$ in F do
        $T \leftarrow L_i\ (K \setminus R_i)$ ;
        test ← true;
        for each J in $\mathcal{K}_S$ do
            if T includes J then test ← false ;
        if test then $\mathcal{K}_S \leftarrow \mathcal{K}_S \cup \{ \mathrm{Key}(\Omega , F, T)\}$
    end
end;
return $\mathcal{K}_S$
```

[+/] Key (Ω, F, X) corresponds to the algorithm Minimal Key in [3] , which determines a key for S that is a subset of a specified superkey X.

The following simple remarks [7] , in some cases can be used to improve the performance of the algorithm of Lucchesi and Osborn.

Remark 3.

To find the first key for $S = <\Omega , F>$, instead of Ω it is better to use the superkey $(\Omega \setminus R) \cup (L \cap R)$ and algorithm I in [I] , and instead of the algorithm Key(Ω, F, T) it is better to use algorithm 2 in [I] for finding one key for S included in a given superkey T .

Remark 4.

In [I] it is shown that

$$R \setminus L \subseteq \Omega \setminus H ,$$

i.e. $R \setminus L$ consists only of non- prime attributes.

Therefore, if $R_i \subseteq R \setminus L$ then

$R_i \cap K = \emptyset$, $\forall K \in \mathcal{K}_S$ and $L_i (K \setminus R_i) \supseteq K$.

That means , when computing $T = L_i (K \setminus R_i)$, we can neglect all FDs $(L_i \longrightarrow R_i)$ with $R_i \subseteq R \setminus L$ for every $K \in \mathcal{K}_S$.

Let us denote

$$\overline{F} = F \setminus \{ L_j \longrightarrow R_j \mid L_j \rightarrow R_j \in F \text{ and } R_j \subseteq R \setminus L \}$$

Remark 5.

With a fixed K in $\mathcal{K}_S$, it is clear that if $K \cap R_i = \emptyset$ then $L_i (K \setminus R_i) \supseteq K$.

In that case , it is not necessary to continue to check whether T includes J for each J in $\mathcal{K}_S$.

So, it is better to compute T by the following order

$$T = (K \setminus R_i) \cup L_i$$

Remark 6.

The algorithm of Lucchesi and Osborn is particularly effective when the number of keys for $S = \langle \Omega, F \rangle$ is small. No, basing on what information one can conclude that the number of keys for S is small ?

There is no general answer for all cases, and it is shown in [8] that the number of keys for a relation scheme $S = \langle \Omega , F \rangle$ can be factorial in $|F|$ or exponential in $|\Omega|$, and that both of these upper bounds are attainable.

However, it is shown in [I , corollary I] that

$$|\mathcal{K}_S| \leq C_h^{[h/2]}$$

where h is the cardinality of $L \cap R$.

Thus, if $L \cap R$ has only a few elements then it is a good criterion for saying that S has a small number of keys.

In the case $L \cap R = \emptyset$, $\Omega \setminus R$ is the unique key for $S = \langle \Omega , F \rangle$ as pointed out in [I , corollary 4] .

Example 2.

Once more, we return to the second example in [3, Appendix I]

$$\Omega = \{a, b, c, d, e, f, g, h\}$$

$$F = \{ a \to b, c \to d, e \to f, g \to h \}$$

It is clear that for this relation scheme

$$L \cap R = \emptyset ,$$

and it has exactly one key, namely aceg.

Taking the remarks 3 - 5 into account, the algorithm of Lucchesi and Osborn now can be presented as follows:

ALGORITHM OL2. Set of all keys for $S = \langle \Omega , F \rangle$;

$\mathcal{K}_S \leftarrow \{$ Algo.I $(\Omega , F, (\Omega \setminus R)(L \cap R))\}$ +/ ;

+/ Algo.I and Algo.2 refer to Algorithm I and Algorithm 2 in[I] respectively .

```
for each K in $\mathcal{K}_S$ do
    for each FD $(L_i \to R_i)$ in $\bar{F}$ such that $K \setminus R_i \neq K$ do
        $T \leftarrow (K \setminus R_i)\ L_i$ ;
        test $\leftarrow$ true ;
        for each J in $\mathcal{K}_S$ do
            if T includes J then test $\leftarrow$ false ;
        if test then $\mathcal{K}_S \leftarrow \mathcal{K}_S \cup \{ \text{Algo.2}(\Omega, F, T) \}$
    end
end;
return $\mathcal{K}_S$ .
```

Acknowledgment

The authors thank Vu Duc Thi for reading one earlier version of the paper and providing useful comments.

References

I HO THUAN - LE VAN BAO :Some results about keys of relational schemas . Acta Cybernetica, Tom. 7, Fasc. I, Szeged, I985, pp. 99-II3.

2 DEMETROVICS, J. - HO THUAN - NGUYEN XUAN HUY - LE VAN BAO : Balanced relation scheme and the problem of key representation . Közlemények MTA - SZTAKI , 32/I985, Budapest, pp. 5I-80.

3 LUCCHESI,C.L. - OSBORN, S.L. : Candidate keys for relations . Journal of Computer and System sciences, I7, I978, pp. 270-279.

4 ULLMAN, J.D. :Principles of database systems. Computer Science Press, Second edition, I982.

5 ARMSTRONG, W.W. : Dependency structures of database relationships. In : Information Processing 74, North Holland Publishing Company, I974, pp. 580-583.

6 FERNANDEZ, M.C. : Determining the normalization level of a relation on the basis of Armstrong's axioms. Computers and Artificial Intelligence, 3 / I984 /, pp. 495-504.

7 HO THUAN : Some remarks on the algorithm of Lucchesi and Osborn . Közlemények MTA - SZTAKI, 35/ I986, Budapest / to appear /.

8 OSBORN, S.L. : Normal forms for relational databases. Ph.D. Dissertation, University of Waterloo, I977.

NATURAL LANGUAGE QUERY FOR DATABASES

Tudor Toma and Elena Saftoiu,
Institute for Computer Technique and
Informatics, Bucharest, Romania

Abstract

In the first section the paper points out the duality of the natural query language and the difference between natural language processing in AI programs and in database interfaces. In view of this difference, some features of NL query processing are presented. The second section is concerned with the analysis of NL queries, based on relational data models. First the premises for simple query analysis are outlined. Then a model for lexical, syntagmatic and semantic processing of these queries is discussed. The third section presents the SINAL system, a framework for implementing stand-alone interfaces in natural language for dBASE III databases. Specific implementation aspects and system performances are mentioned.

1. The duality of natural query language

Natural language understanding, as an AI field, deals with the analysis and representation of natural language in complex structures, as abstract models of the real world. As pointed out in [3], the models used for database structures, called "data models", are simpler then those used in NL understanding systems. On the other hand, NL interfaces based on data models usually deal with a very large amount of data. Consequently, NL query processing presents specific features:

a. The NLQ interfaces are not concerned with the understanding of the complex meaning of the NL input but rather with the translation from NL form into the formal database interface language. The query meaning is tightly correlated with the database data model.

b. Generally, database users formulate simple NL queries and access data in small successive steps. Short queries are a feature of the NL itself and assure a higher data accuracy.

c. In order to access a database defined by a rigurous data model, the user´s language is restricted to an extent necessary for the usage of the system. Under these conditions, natural query language still remains a formal language[3].

d. NLQ interfaces, like any friendly interface program, must take into account a set of dialog engineering rules[1]:

- avoid acausality: make the activity of the system a clear consequence of the user´s actions;
- uniformity and consistency: ensure that all terminology and semantics are uniformly available and consistently applied throughout all interface activity;
- make the state of the dialog observable:

the response should be sufficient to identify the type of the current activity.
- validate data on entry by checking syntax and values. Have the user himself revalidate major updates before acting upon them.

The usual NL queries could be considered as expressed in a "formal" natural language. Because of this dual nature, there are various and contradictory opinions about the success of NL interfaces for database systems[3],[4]. This paper illustrate a simple and versatile approach to NLQ, which covers the range from fully natural laguage query to formal query facilities.

2. A model for understanding NL queries

Our model for understanding natural language queries performs a translation of the input query into an intermediate form. The model is dedicated to relational database query and not to general NL analysis. Consequently, as discussed in the previous section, the NL input, practically not restricted from lexical and syntactical points of view, must be formulated according to a query semantic grammar. The grammar accepts compound queries, every simple query being composed of a command and some of its structured attributes, given in an arbitrary order.

2.1. Premises for simple NL query processing

The relational database systems are based on a relational data model which accepts queries with a simple internal structure. Besides the command, simple queries may include a set of (structured) elements as the field list, the filter, the peripheral list, the entity, the command execution options, etc. The data model provides some particular features for the NL query processing :

- the words can be generally represented in the dictionary in stem form. The expensive morphemic analysis is not necessary.
- the lexical processing becomes practically language independent and makes the entire query analysis language independent.
- the NL input is not subject to a global syntactic analysis. The syntactic analysis processes only local syntactic aspects permitting a great liberty for the relative position of semantic elements within the NL query.
- the semantic analyzer is fast and accurate. It provides a large variety of facilities such as references to previous queries or query elements, the developement of the referenced query elements, the use of default queries or of default query elements.
- an easy diagnose of incorrect queries and the evaluation of query ambiguities.
- the lexic can be extended with simple symbols introduced in the dictionary as synonyms for usual, application-dependent syntagms. These symbols provide for an abreviated query language and a great versatility in dealing with database applications, for users with increasing

2.2. The query analysis

Three levels of analysis, lexical, syntagmatic and semantic are performed, each level using specific representation and processing techniques.

At the lexical level all the items of the input query are mapped in the dictionary. The dictionary contains either word stems or words in complete form, if necessary, in order to avoid lexical ambiguity. The items found in the dictionary form the output list, the others are discarded. Synonyms are solved at this level.

The syntagmatic analysis is performed by a rule driven ascendant parser. The parser transforms the lexical analysis output into a list of terminals of the query semantic grammar. The rule set do not define a formal input language. The rule driven analysis performs mostly syntagmatic processing and is focused on small parts of the processed list, so its effects are local. The parsing becomes more accurate as the rule set increases in complexity.

At the semantic level some structured elements need their information to be grouped, so that a local syntactic analysis is indispensable.

The semantic grammar is represented by a semantic network based model called a representation network. The input list is processed in the representation network by local, node oriented procedures activated by a control algorithm. If the input is semantically correct, it is transformed into an intermediate form. Otherwise, the analyser asks for the reformulation of the missing semantic elements.

3. SINAL A natural language interface for dBASE III databases

This section presents a frame system for implementing interfaces for dBASE III databases in natural language. The dBASE III interfaces are implemented with SINAL as stand-alone programs which interfere with a database application only by its data files. The interfaces accept the general dBASE multifile multientity database structure and all the dBASE commands, including the access to a large variety of coded fields. The application database may be created and modified either with the dBASE system or using a special editor and a set of data management programs written in dBASE, which extend the system facilities in order to fit the SINAL interface. The extended dBASE framework and the SINAL interface, designed as an integrated package, provide a performant tool for creating, manageing and friendly accessing dBASE applications.

SINAL is composed of two basic parts: the preprocessor and the natural language interface processor.

3.1. The preprocessor

This first module performs a preliminary processing of the domain-dependent information, the dictionary and the syntagmatic rules. This information is edited into two text files and preprocessed into a couple of coded files. The program analizes the domain-dependent information consistency and ensures a minimum storage form for the coded output data.

The lexical and syntagmatic analysis are based on the domain-dependent information preprocessed in the coded files. During the coding process, rule activation information is attached to the corresponding lexical items. These items, when present in the processed list, guide the syntagmatic analysis by activating only thoses rules that may successfully match in the given context.

For every particular dBASE application, a specific lexicon must be appended to the system dictionary. Rules must be added to model some low level syntactic aspects and syntagms of the application query language.

3.2. The interface processor

The interface processor consists of two main modules and two auxiliary modules. The first module is a query analyzer which implements the query understanding model for dBASE III databases and performs the query translation into intermediate code form. The second module is a query interpreter which executes the intermediate code commands. It eliminates the dBASE critical time overhead and internal buffer restrictions, providing a highly performant access to the dBASE III datafiles, through the stand-alone SINAL interface. The auxiliary modules implement the query management and a menu-driven help.

3.2.1. The query interpreter

The query interpreter loads the information from the database files, using an adequate memory allocation strategy, selects the information according to the command filter and performs the command action.

The structure loader is application independent but depends on the data model of the database system, in this case the dBASE relational model. At the beginning of the query session, the structure loader reads from a directory file the names of the application data files (dBASE relations), accesses their structure block and creates an internal representation, as one equivelent extended relation. This internal data model requires the existence of a common key field in each data file and removes the retrieval restrictions imposed by dBASE, which limits the number of active data files. The model provides a versatile storage for the entity characteristics; for each entity, attribute sets can be omitted or introduced with multiple values; it is also possible to surpass the dBASE system facilities by simulating variable length fields for databases developed with the SINAL associated editor mentioned at the beginning of this section.

During the query session, for a given request the data loader activates all the associate data files, which are completely or partialy loaded. The data buffer is adapted to the database size, at the beginning of the query session.

The interpreter selects the requested information according to the intermediate code filter. The filter expression accepts the usual arithmetic and logic operators, a substring search operator and unrestricted parantheses levels. The interpreter performs the selection and projection operations and a set of usual functions (count, total, min, max, average, etc); current work is dedicated to the implementation of the complete set of relational operators. Simple coded fields, code list fields, fields with codes grouped by code set tags and codes with subcode patterns are decoded before output. For all the coded fields, application dependent output patterns can be implemented.

The query result can either be just displayed on the screen or structured as a report. The output may be simultaneously displayed, printed and stored on disk.

3.2.2. The auxiliary modules

The query management module permits to associate NLQ queries with tags, to store and to delete queries, to invoke them by their tags and to display stored tags and their associated information. Sets of query tags can be attached to new tags, implementing a hierarchical metaquery level, useful for frequent routine query sequences.

The HELP menu-driven module provides information about the database structure, the query semantic grammar, the query control facilities and the query management system.

3.3. The system implementation

SINAL is written in "C" on an IBM-PC compatible microcomputer in a MS-DOS environment. It was designed and implemented to assure a fast query response, a fully natural query language and a great flexibility and portability. All searches are performed by hashing. The rule-based analysis is efficiently bottom-up driven. The semantic processing control is information gain driven providing a minimal search in the representation network.

The representation network model and all the AI processing methods and control techniques used for the query analyzer were modeled and tested in LISP prior to their implementation in "C".

The query analysis is performed immediately. The query execution depends on the average disk access time and on the number of necessary accesses given by the database structure and size, and by the query complexity. For medium databases and for intensely coded applications all the information can be loaded in the interface internal buffer, providing an immediate query response.

The first interface application was implemented for a database on personal and professional computers, in both Romanian and English languages. The second application is a NLQ interface for a medical database now used in a radiobiology clinic.

References

[1] B.R.Gaines,M.L.G.Shaw: Dialog engineering, Design for human - computer communication, Academic Press, N.Y. (1983)

[2] I.D.Hill: Natural language versus computer language, Design for human - computer communication, Academic Press, N.Y. (1983)

[3] K.D.Krageloh,P.C.Lockemann: Access to data base systems via natural language, Lecture notes in computer science 63, Springer Verlag, (1978)

[4] D.Tufis,D.Cristea,G.Ciobanu: QUERNAL - a natural language interface for databases, The II-nd National Symposium of AI, Bucharest (1985)

1986-BAN EDDIG MEGJELENTEK:

179/1986 Terlaky Tamás: Egy véges criss-cross módszer és alkalmazásai

180/1986 K.N. Čimev: Separable sets of arguments of functions

181/1986 Renner Gábor: Kör approximációja a számitógépes geometriai tervezésben

182/1986 Proceedings of the Joint Bulgarian-Hungarian Workshop on "Mathematical Cybernetics and Data Processing" Scientific Station of Sofia University, Giulecica /Bulgaria/, May 6-10, 1985 /Editors: J. Denev, B. Uhrin/ Vol I

183/1986 Proceedings of the Joint Bulgarian-Hungarian Workshop on "Mathematical Cybernetics and Data Processing" Scientific Station of Sofia University, Giulečica /Bulgaria/, May 6-10, 1985 /Editors: J. Denev, B. Uhrin/ Vol II

184/1986 HO THUAN: Contribution to the theory of relational databases

185/1986 Proceedings of the 4th International Meeting of Young Computer Scientists IMICS'86 /Smolenice, 1986/ /Editors: J. Demetrovics, J. Kelemen/

186/1986 PUBLIKÁCIÓK - PUBLICATIONS 1985
Szerkesztette: Petróczy Judit

187/1986 Proceedings of the Winter School on Conceptual modelling /Visegrád, 27-30 January, 1986/ /Editors: E. Knuth, A. Márkus/

188/1986 Lengyel Tamás: A Cluster analizis néhány kombinatorikai és valószinüségszámitási problémája

189/1986 Bernus Péter: Gyártórendszerek funkcionális analizise és szintézise

190/1986 Hernádi Ágnes: A tipus fogalma, és szerepe a modellezésben

191/1986 VU DUC THI: Funkcionális függőséggel kapcsolatos néhány kombinatorikai jellegü vizsgálat a relációs adatmodellben

192/1986 Márkusz Zsuzsanna: Papers on Many-stored logic as a tool for modelling

193/1986 KNVVT Conference on Automation of Information Processing on Personal Computers
Budapest, May 5-9, 1986 Vol I.
Szerkesztette: Ratkó István