KNOWLEDGE REPRESENTATION OF DESIGN IN MANY-SORTED LOGIC

Zsuzsanna Markusz

Computer and Automation Institute
Hungarian Academy of Sciences

ABSTRACT

The paper presents a formal tool of modelling of constructional design, where the cardinal issue is knowledge representation. It is shown that the structures of classical mathematical logic, such as t-type many-sorted models, can be applied almost directly to represent the world of a certain designing task. Using the PROLOG programming system, the logical representation of the task yields, at the same time, its automated solution. The most important moment of modelling a task of this kind is fixing the appropriate type of the structures. A PROLOG program for designing many-storied buildings is taken as an example.

I INTRODUCTION

The aim of this paper is to show a logic-based method for modelling certain tasks of constructional design which helps to develop ready-to-run Computer Aided Design (CAD) systems. The main idea of this method is the application of t-type many-sorted models of classical mathematical logic to represent designing problems. The final form of the logical representation is written in a very high level language called PROLOG, which can be directly run on a computer.

The problem-solving method described below is the generalization of the experience gained by solving a few technical designing tasks in PROLOG [5, 6, 7, 8j. We shall illustrate the application of this method on a concrete architectured CAD task, namely designing many-storied apartment houses.

II MANY-SORTED MODELS

A language L of the classical mathematical logic is a triple

$L_t = \langle F_t, M_t, \models_t \rangle$

where F_t is the syntax (set of well-formed formulas)

M_t is the class of models and

$\models_t$ is the validity relation. ($\models_t \subset M_t \times F_t$).

The models are relational structures of a fixed type. The type t is a pair of functions,

$t = \langle t_R, t_F \rangle$ where

t_R and t_F define the arity of the relation and function symbols, respectively. See [1].

The many-sorted models are modification of these t-type models. The fundamental difference between many-sorted models and t-type models is that the universe of the many-sorted models is not homogeneous but consists of disjunct sets of different sorts. Thus, when defining the types of functions and relations, we have to give not only the number of the arguments but the sort of every argument as well.

Let $S = \{\lambda_i : i<\omega\}$ be the set of sorts of the elements of the universe.

Then the universe of the many-sorted model is

$A = \{ A\lambda_i, \lambda_i \in s \}$

The type (signature) of the many-sorted model $\mathfrak{A}$ (see in [3])

i) $S = \{\lambda_i \; i<\omega\}$ the set of sorts;

ii) to every n-ary function symbol f_n, $n<\omega$ a pair $\langle f_n, \langle \lambda_1, \dots, \lambda_n ; \lambda_{n+1} \rangle\rangle$ is assigned,

where f_n is the function symbol, $\langle \lambda_1, \dots, \lambda_n ; \lambda_{n+1} \rangle$ the sequence of sorts.

$\lambda_1, \dots, \lambda_n$ are the sorts of the arguments of the function, and λ_{n+1} is the sort of the value of the function.

iii) to every n-ary relation symbol r_n, $n<\omega$ a pair, $\langle r_n, \langle \lambda_1, \dots, \lambda_n \rangle\rangle$ is assigned,

where r_n is the relation symbol

$\langle \lambda_1, \dots, \lambda_n \rangle$ is the sequence of the sorts of the arguments of the relation.

Thus the t-type of a many-sorted model is a triple

$t = \langle S, t_R, t_F \rangle$,

where S is the set of sorts,

t_F is the type-function of the functions defined in ii)

t_R is the type-function of the relations defined in iii)

The syntax of many-sorted languages is basically the same as that of the classical ones, with the following exception: The signature determines the sort of terms occuring as arguments of the relation or function symbols.

III MODELLING OF CONSTRUCTIONAL DESIGNING TASKS

Let us consider a technical designing task whose aim is to create a complex construction using basic units, according to a given set of rules: e.g. to compose many-storied houses of apartments or machines of spare parts. The basic units are represented by constants, while the partial constructions and the final construction by compound expressions. Thus we can regulate the hierarchy that will constitute the final construction by defining the type-function t of the t-type model.

We can attach conditions to every step of making the construction, i.e. the compound expression representing it. The co ection between the parts of the construction is represented by relations. If in every stage of the designing process all requirements of the construction are given by the corresponding relations, then, in fact, the necessary "knowledge" is given, i.e. "the world of the design" is defined. The choice of these relations is but the definition of the type function t . As a result, **we can declare that the most important moment of modelling a constructional designing task by classical many-sorted t-type structures is fixing the appropriate type $t = \langle S, t_R, t_F \rangle$.**

Let us go back to describing the world of the designing task. We aim not only at the representation of all requirements of the technical construction by mathematical relations, but also to create the given product with a computer system. To achieve this, we need to express the requirements in a spe-cific form which can be used by an automatic mechanism. Since this specific form is nothing else than formulas of logic, the mechanism mentioned should be a certain kind of calculus. PROLOG is a computer system based on first-order logic where the running of a program is controlled by a complete calculus present in the interpreter.

A PROLOG interpreter accepts Horn-clauses, i.e. implications whose conditions are connected by the logical connective "and", and have only one conclusion. In this paper we do not wish to give a detailed description of PROLOG. See [10, 11].

Note that PROLOG is based on a one-sorted language. However, it does not make any difference, because the pattern matching mechanism built in the deduction system of PROLOG automatically fulfils the requirements that only the terms of the corresponding sorts should be substituted for each other.

IV APPLICATION OF MANY-SORTED LOGIC IN MODELLING OF THE DESIGN OF MANY-STORIED HOUSES

IV.1. The task

The task is to work out a model of the designing process of many-storied apartment houses in PROLOG which will result in a ready-to-run CAD system. Our purpose is to design one or more houses, so the different customer can choose from different ground-plans, and at the same time these buildings will meet certain technical and functional requirements.

One program will provide a set of different ground-plans for every customer who can set a priority order excluding the less good ones. Another program will design the whole building, placing the apartments beside and above one another. We can get several variants of buildings by changing the input data of the building given by the architect and/or by reordering or extending the computerized representation of the apartment variants stored in a disc file. The detailed description of the program system can be found in [6]

The programs are run on an IBM 3031 computer of the Hungarian Academy of Sciences; the PROLOG system was installed by Peter Kdves. The results of our PROLOG program was graphically represented on a GD80 graphic display [4] by lAszl6 Andor and on a CALCOMP plotter by Judit Takacs.

A three-storied apartment house designed by the program is shown in Figs. 2 and 3.

In order to be able to present the logical modelling of the task, we have to get acquainted with the **architectural conception** determining our model.

The basic elements of an apartment are the cells with different sizes and functions. According to their size the cells can be classified into two groups: narrow and wide cells. The middle of the apartment is constructed by joining three wide cells, whereas the perimeter is made up by 2-4 narrow cells. An apartment consists of the middle and the outside parts together (Fig. 1).

A narrow or a wide cell is defined by its function and its variation number. The function of a cell can be the following: living room, bedroom, kitchen, dining-room, entrance, water-block (bathroom + toilet). The variation numbers of the narrow and wide cells define the number and locations of the doors in the cells.

The basic units of a building are the apartments. Two apartments and a staircase make one section. The horizontal linking of one or two sections make one level. The vertical linking of 2-4 levels makes one unit. Any number of units linked horizontally make up one building.

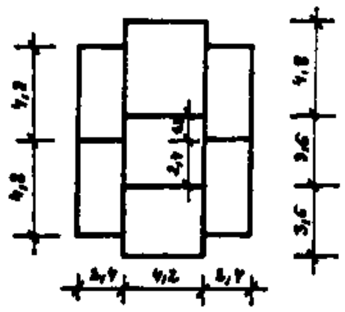


Figure 1.
Outline of an apartment

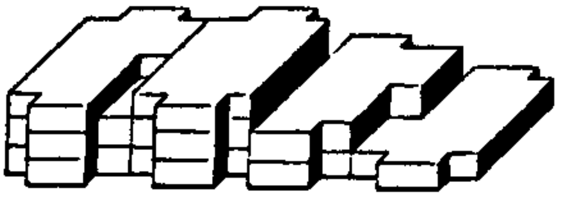


Figure 2. Perspective scheme of a building designed by the program

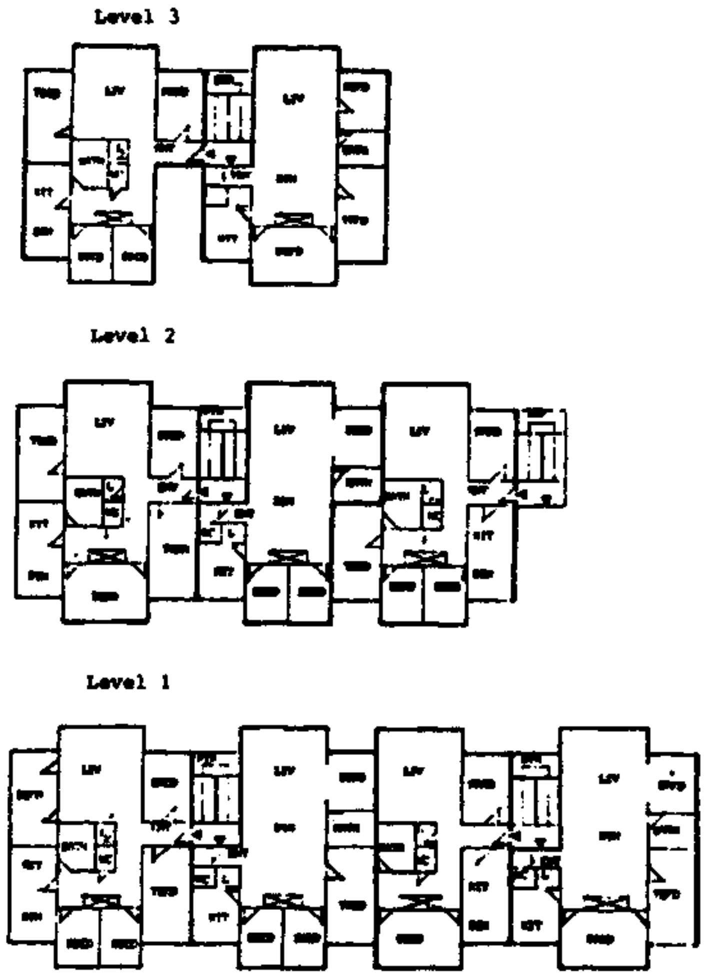


Figure 3. The level-by-level ground-plan arrangement of the building in Fig. 2.
Abbreviations: LIV: living room - KIT: kitchen - DIN: dining room - BATH: bathroom - DBED: double bedroom - TBED: twin bedroom - SBED: single bedroom - L: larder - ENT: entrance - STR: staircase

IV.2. The logical representation of the task

IV.2.1. Sorts

$S = \{f, vn, vw, nc, wc, o, m, a, s, l, u, b, \omega\}$

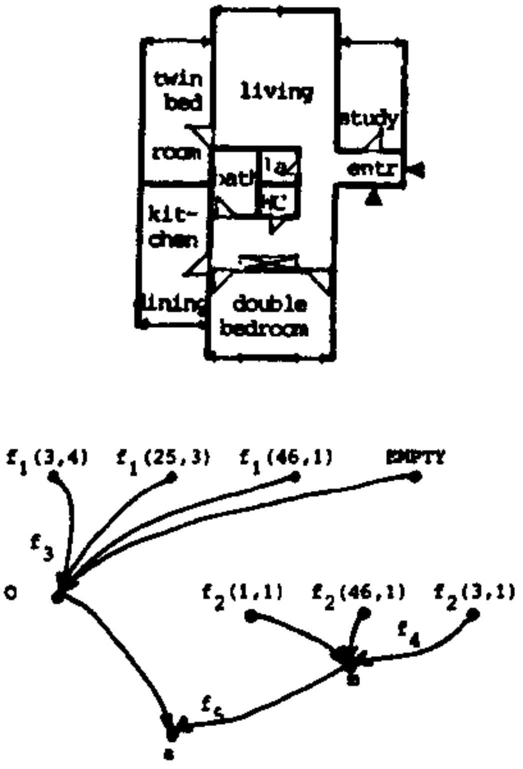
where

f	- the function of a basic unit, i.e. a cell
vn	- variation of a narrow cell,
vw	- variation of a wide cell,
nc	- narrow cell,
wc	- wide cell,
o	- the outside of an apartment,
m	- the middle of an apartment,
a	- apartment,
s	- section,
l	- level,
u	- unit,
b	- building,
ω	- non-negative integers.

IV.2.2. Functions

The set of function symbols is as follows:

$\Phi = \{f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9\}$.



$f_5(f_3(f_1(3,4), f_1(25,3), f_1(46,1), \text{EMPTY}), f_4(f_2(1,1), f_2(46,1), f_2(3,1)))$

Figure 4. An apartment and the compound term representing it

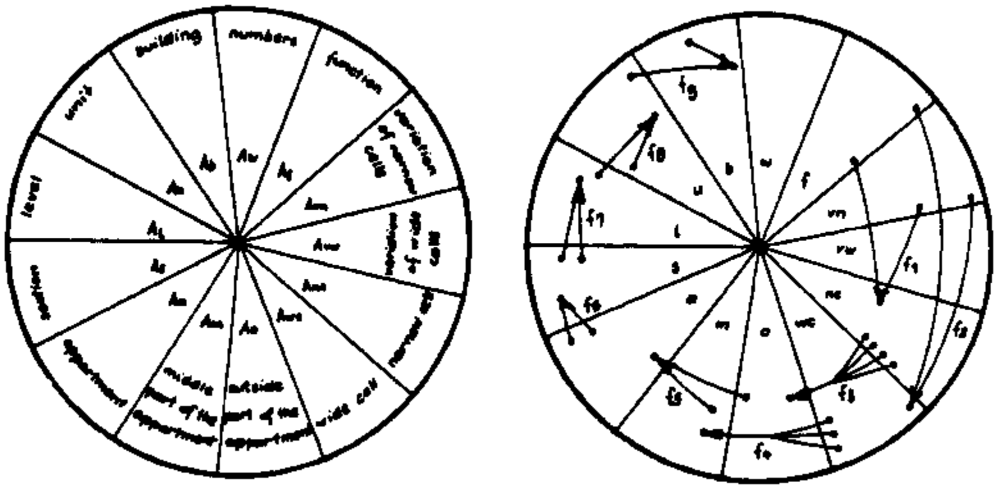


Figure 5. Many-sorted functions

The type-function t_F is a set of pairs:

$\langle f_1, \langle f,\ vn;\ nc\rangle\rangle$	$\langle f_5, \langle m,\ o;\ a\rangle\rangle$
$\langle f_2, \langle f,\ vw;\ wc\rangle\rangle$	$\langle f_6, \langle a,\ a;\ s\rangle\rangle$
$\langle f_3, \langle nc,\ nc,\ nc,\ nc;\ o\rangle\rangle$	$\langle f_7, \langle s,\ s;\ l\rangle\rangle$
$\langle f_4, \langle wc,\ wc,\ wc;\ m\rangle\rangle$	$\langle f_8, \langle l,\ u;\ u\rangle\rangle$
	$\langle f_9, \langle u,\ b;\ b\rangle\rangle$

We shall illustrate the functions which construct the whole building from apartments (Fig.5.).

$\langle f_6, \langle a, a; s\rangle\rangle$

Two apartments and one staircase together make up one section.

$\langle f_7, \langle s, s; l\rangle\rangle$

The horizontal linking of two sections makes up one level.

$\langle f_8, \langle l, u; u\rangle\rangle$

The vertical linking of a level and a unit makes up a new unit.

$\langle f_9, \langle u, b; b\rangle\rangle$

By linking a unit and a building, we can get a new building. To construct a one-unit building, we use the "empty building". (Fig. 6.)

IV.4. Relations

As an example, we examine the relations between the apartments which determine the conditions of constructing the buildings in a proper way.

The respective part of the type-function t_R is the set of pairs:

$\langle S, \langle a,\ a\rangle\rangle$	$\langle V, \langle a,\ a\rangle\rangle$
$\langle ST, \langle a,\ a\rangle\rangle$	$\langle SU, \langle a,\ a,\ a\rangle\rangle$
$\langle HZ, \langle a,\ a\rangle\rangle$	$\langle GL, \langle \omega,\ a,\ a\rangle\rangle$
$\langle SL, \langle \omega,\ a\rangle\rangle$	$\langle GR, \langle \omega,\ a,\ a\rangle\rangle$.
$\langle SR, \langle \omega,\ a\rangle\rangle$	

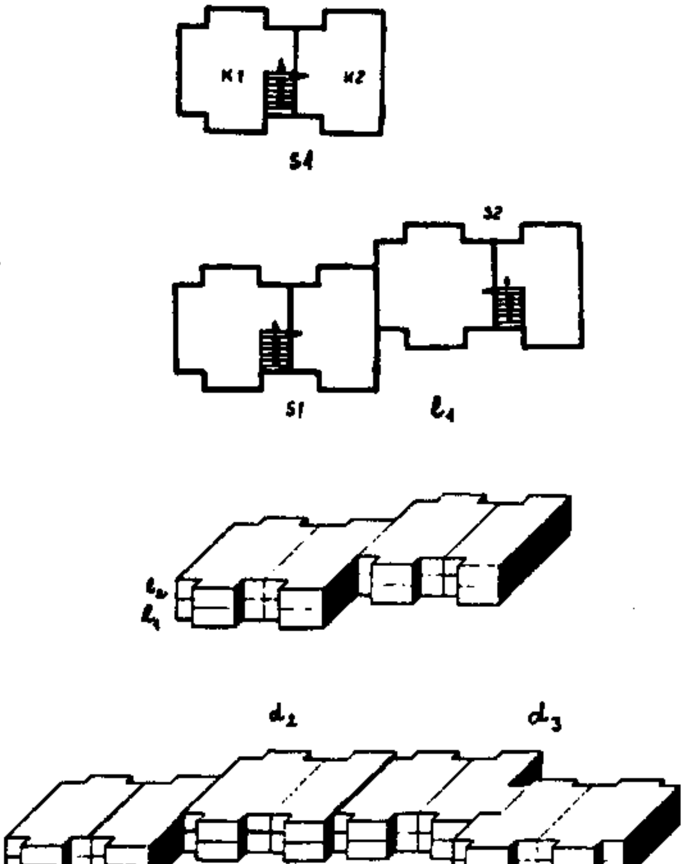


Figure 6. A building consisting of two units

These relations not only define connections, but as parts of the PROLOG program they are procedures with controlling or construction function. Let us examine them one after another.

1. Symmetry: $S(X, K_{12})$

Valid if apartment K_{12} is the mirror image of apartment X with respect to the y-axis.

2. Staircase: $ST(K_{11}, K_{12})$

Valid if the staircase can be adjusted between two apartments next to each other (K_{11}, K_{12}), so that the entrances will open from the staircase.

3. Horizontal: $HZ(K_{12}, K_{13})$

Valid if the geometrical structure of the right segments of the first apartment (K_{12}) and that of the left segments of the second apartment (K_{13}) are capable to join each other.

4. and 5. Suitable left (suitable right): $SL(0, K_{11})(SR(1, K_{14}))$

Valid if the left (right) segments of apartment (K_{11}) (K_{14}) on the first level, i.e. on the left (right) side of the unit, correspond to the left (right) type of the unit.

6. Vertical: $V(K_{11}, K_{21})$

Valid if the water-block, the entrances and the segment of the staircase of the upper apartment (K_{21}) and that of the apartment below it (K_{21}) are on top of one another.

7. Support: $SU(K_{21}, K_{11}, K_{12})$

Valid if there are lower segments under each segment of the upper apartment (K_{21}).

8. and 9. Geometric left (geometric right): $GL(0, K_{11}, K_{21})(GR(1, K_{14}, K_{24}))$

Valid if the left (right) segments of the apartment correspond to the type of the left (right) side of the unit, and the geometry of the upper and lower apartments match.

Having defined the type of the language, the axioms written in that language constitute the specification of the designing problem. These axioms define the above functions and relations, and so they determine the class of possible models. To obtain a PROLOG program from these axioms, it may be necessary to introduce new functions and relation symbols. Let us use the symbols t'_F and t'_R for these "auxiliary" functions and relations. Thus the definition of the type

$$t = \langle S,\ t_R \cup t'_R\ ,\ \ t_F \cup t'_F \rangle$$

will be completed after finishing the specification.

V. EXAMPLE FOR THE DEFINITION OF A RELATION IN THE PROLOG PROGRAM

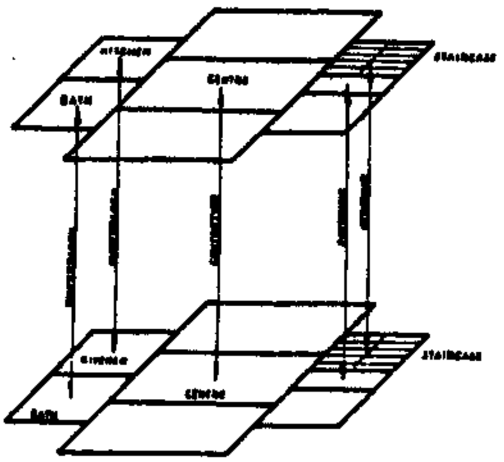


Figure 7. Illustration of the VERTICAL relation of two apartments

```
+VERTICAL(*M1;*F1,*M2;*F2)
      -CENTRETYPE(*M1,*M2,*X)
      -WATERBLOCK(*X,*F1,*F2)
      -ENTERING(*F1,*F2)
      -VERSTAIR(*F1,*F2).

+CENTRETYPE(*Q1.*N.*L1, *Q2.*N.*L2,*N).
+WATERBLOCK(46!*V, *F1,*F2)
      -KITCHEN(*F1,*F2).
+WATERBLOCK(42!*V, *F1,*F2)
      -KITCHEN(*F1,*F2)
      -BATH(*F1,*F2).

+KITCHEN(*F1,*F2)
      -THESAME(*F1,*F2, 45, 25).
+BATH(*F1,*F2)
      -THESAME(*F1,*F2, 63, 63).

+ENTERING(*F1,*F2)
      -EXISTENTER(*F1) - EXISTENTER(*F2)
      -THESAME(*F1,*F2, 43, 45).

+ENTERING(*F1,*F2)
      -EXISTENTER(*F1)
      -NOT(EXISTENTER(*F2)).

+ENTERING(*F1,*F2)
      -NOT(EXISTENTER(*F1)).

+EXISTENTER(*F)
      -MEMBER(*F,*COD!*V)
      -ENTRANCE(*COD!*V).

+VERSTAIR(*F1,*F2)
      -EXISTSTAIR(*F1) - EXISTSTAIR(*F2)
      -UNDER(*F1,*F2, L!L).

+VERSTAIR(*F1;*F2)
      -EXISTSTAIR(*F1)
      -NOT(EXISTSTAIR(*F2)).

+VERSTAIR(*F1,*F2)
      -NOT(EXISTSTAIR(*F1)).
```

```
+EXISTSTAIR(*F1)
        -MEMBER(*F1,*COD!*V)
        -CONSTANT(*COD)
        -EQUAL(*COD!*V, L!L).
```

The relation VERTICAL is needed in the selection of every apartment above the ground floor. It has two arguments,

1. the lower apartment (*M1;*F1) and
2. the upper apartment (*M2;*F2),

and it is true if the following four conditions are fulfilled:

-CENTRETYPE(*M1,*M2,*X)
It checks whether the middle wide cells of the two apartments are the same; if they are, it gives their value to the variable *X.

-WATERBLOCK(*X,*FI,*F2)
The variant *X already contains the value (with or without bathroom) of the middle cell. Depending on this value, the relation checks whether the kitchens or the kitchens plus bathrooms are on top of one another in the two apartments.

-ENTERING(*FI,*F2)
If there is an entrance in each apartment, it checks whether they are on top of one another. (The hall of the middle cell can also be the entrance; in this case there is no entrance in the narrow segments.)

-VERSTAIR(*FI,*F2)
If a staircase belongs to each apartment, it checks whether they are on top of one another.

Note that the definition of the auxiliary relations TRESAME, MEMBER, ENTRANCE, UNDER is withheld here but they are of course defined in the ready-to-run program.

VI CONCLUSION

A new technique for the solution of CAD problems is presented and illustrated on an architectural example. The designing knowledge is represented in the form of axioms described in the language of mathematical logic. The calculi of logic which manipulate these axioms can be used to model designing processes.

In this paper we wish to point out that the decisive step in the above method is the selection of the adequate type of language. If Horn-formulas are used, then the description of the technical problem can be considered as a program written in PROLOG.

VII ACKNOWLEDGEMENT

I would like to express my appreciation to Mr. Miklos Sz6ts for the helpful and inspiring discussions we had about this subject.

REFERENCES

[*I] Andreka, H., Gergely, T., Nemeti, I.: "Easily Comprehensible Mathematical Logic and its Model Theory." Publication of the Central Research Institute for Physics. Budapest, Hungary. (1975)

[2] Chang, C.C., Keisler, H.J.: "Model Theory." North-Holland Publishing Company, Amsterdam, The Netherlands (1973)

[3] Gergely, T., Szots, M.: "Logical Foundation of Problem Solving." Proceedings of II. IMAI, Leningrad, Repino, USSR 1980)

[4] Hatvany, J., VerebelY, P.: "Distributed Intelligence in the GD80 Display System." SID Symposium, Chicago (1979)

[5] Markusz, Zs.: "How to Design Variants of Flats Using Programming Language PROLOG Based on Mathematical Logic." Proceedings of IFIP'1977, Toronto, Canada (1977)

[6] Markusz, Zs.:"Application of PROLOG in Designing Many-Storied Dwelling Houses."Preprint of Logic Programming Workshop, Debrecen, Hungary. Edited by Sten-Ake TArnlund, University of Stockholm, Sweden (1980)

[7] Markusz, Zs.: "Logic Based Programming Methods and their Applications to Solve Architectural CAD problems."(in Hungarian) Ph.D. Thesis, University of Sciences, Budapest, Hungary (1980)

[8] Santha, E.: "Application of PROLOG in Hungary." Preprint of Logic Programming Workshop, Debrecen, Hungary. Edited by Sten-Ake Tarnlund, University of Stockholm, Sweden (1980)

[9] Szeredi, P., Put6, I.: "PROLOG Manual." Szamologep, 7. 3-4. sz. (in Hungarian) (1977)

[10]Warren, D.:"Implementing PROLOG - Compiling Predicate Logic Programs." Dept. of Artificial Intelligence, Edinburgh University (1977)