Representation and Processing of Structural Design Codes

Représentation et élaboration des normes de construction en génie civil

Darstellung und Verarbeitung von Normen des Bauwesens

Vlasis K. KOUMOUSIS
Assoc. Prof.
Nat.Techn. Univ. Athens
Athens, Greece

Vlasis Koumousis graduated in civil engineering at NTUA in 1975, and received his M.Sc. and Ph.D in applied mechanics from Polytechnic University of New York in 1976 and 1980 respectively. He has worked as consultant and currently is an associate professor at NTUA.

Panos G. GEORGIOU
Ph.D. Student
Nat.Techn. Univ. Athens
Athens, Greece

Panos Georgiou graduated in civil engineering at NTUA in 1988 and currently is working on his doctoral thesis. His research is involved with technical code based optimal design of multi-storey buildings. He has also worked in consulting firms designing reinforced concrete buildings.

Charis J. GANTES
Assist. Prof.
Nat.Techn. Univ. Athens
Athens, Greece

Charis Gantes graduated in civil engineering at NTUA in 1985, and received his M.Sc. in Civil and Ph.D in structural engineering from MIT in 1988 and 1991 respectively. Since 1994 he is assistant professor at NTUA. His research interests are in computer-aided structural design.

SUMMARY
A logical scheme is presented for the representation and processing of structural design codes. The subdivision of codes into several distinct, strongly interrelated parts, and the frequent cross-references between different codes make it difficult to grasp the structure and the domain of applicability of different provisions. The declarative nature of the code provisions and the sequence of their demands can be implemented directly in logic programming. Moreover, the search for solutions can be performed with the sequential backtracking algorithm of Prolog language, while the design space can be formed using several relational databases accommodated in Prolog. The proposed scheme is illustrated with the design of a steel roof using Eurocode 3.

RÉSUMÉ
Un schéma logique est proposé pour la représentation et l'utilisation de normes de construction en génie civil. La subdivision de ces normes en plusieurs chapitres, très interdépendants, et les fréquentes corrélations entre ces normes rendent difficile la compréhension de la structure et le domaine d'application des différentes règles. La nature descriptive des règles et la succession des prescriptions peuvent être mises en oeuvre directement par un logiciel. La recherche de solutions peut être réalisée en utilisant les algorithmes écrits en langage Prolog. Le schéma proposé illustre le projet d'une toiture métallique utilisant l'Eurocode 3.

ZUSAMMENFASSUNG
Beschrieben wird ein logisches Schema für die Darstellung und Verarbeitung von Normen des Bauwesens. Durch die Unterteilung der Normen in verschiedene, miteinander verbundene Teile und wegen der vielen Querverweise zwischen verschiedenen Normen ist es sehr schwer, die Struktur und den Bereich der Anwendbarkeit einzelner Bestimmungen zu verstehen. Der deklarative Charakter der Bestimmungen und die Reihenfolge ihrer Forderungen können aber direkt logisch programmiert werden. Insbesondere kann die Suche nach Lösungen mit der Programmiersprache Prolog durchgeführt werden, während das Lösungsgebiet unter Benutzung verschiedener relationaler Datenbanken auch in Prolog formuliert werden kann. Das vorgeschlagene Schema wird für den Entwurf eines Stahldaches mit Eurocode 3 erläutert.

1. INTRODUCTION

This work summarizes the general philosophy that governs modern structural codes, and outlines the major steps towards the computerized representation of codes together with their incorporation in automated structural design software. The complex structure of design codes and the frequent cross-references in them make it difficult for the users to follow the line of reasoning and establish the interconnection between interrelated specifications. A similar situation exists also within an integrated computer system that contains analysis and design modules. Generally, codes contain both qualitative information and algorithmic procedures to perform various conformance checks, or the design of specific members. Therefore, different means are used to represent these two distinct types of information. A general scheme based on logic programming is presented for the interpretation of the algorithmic requirements imposed by structural codes, as a part of a structural optimization process. This approach is illustrated with specific clauses that describe some of the requirements of "Eurocode No. 3" [1] for the design of steel structures.

2. GENERAL PHILOSOPHY OF STRUCTURAL CODES

As opposed to older structural codes, that were based on the allowable stress concept, most of today's codes follow the principles of ultimate limit state design [2]. This means that the structure can exhibit plastic deformations under extreme loading conditions. Moreover, the structure must be designed for specific limit states defined in the code. These limit states are the ultimate limit states, that ensure non-collapse of the structure, or other forms of failure, and the serviceability limit states, that provide a control on the damageability of the structure. Besides the limit states, important features of the codes are the partial coefficients, that replace the overall safety factor of the allowable stress design philosophy. There are the partial coefficients for the loads and partial coefficients for the materials used in the structure. Partial coefficients are important in all the conformance checks required by a code. Such checks follow a specific safety format that usually takes the following form:

$$S_d \leq R_d \tag{1}$$

where S_d refers to the design value of the actions, that are needed for a particular conformance check, and R_d is the design value of the corresponding resistance. Relation (1) can be used either in explicit, or implicit form to calculate a geometric or other parameter of a component of a structure. The design values represent deterministic estimates of the stochastic functions that describe the actions and resistances, and are expressed in terms of their corresponding nominal or characteristic values S_k and R_k in relations of the type:

$$S_d = \gamma_F S_k, \quad R_d = R_k / \gamma_m \tag{2}$$

where γ_F is the partial coefficient for the action, and γ_m the partial coefficient for the material.

The reliability of the produced designs must be proven on the basis of the structural codes that apply for each case. Reliability analysis studies are important to validate structural design codes ([3], [4]). From the existing structural codes very few have been checked for consistency [25].

3. CODE REPRESENTATION

The information contained in a code is traditionally addressed to the designer, who needs to have complete control of the code, so as to fully interact with it during the process of designing a structure. Today, codes constitute also important parts of integrated computer systems, where, depending on the form of their representation, they interact with other software modules towards the integrated design of structures.

Current integrated systems that perform the analysis and design of structures contain a set of routines to carry out the necessary checks, or calculate the required quantities. These are linked to the analysis module to perform the so-called "one shot analysis". The outcome of such an analysis is a conformance check with the algorithmic part of the code. This is the "hard-coding" approach and is the most widely used in existing automated structural design software packages.

To adapt their orientation to the current needs, modern codes tend to provide information in an algorithmic way. Thus, besides the traditional format that expressed the information in the form of design graphs and tables, more and more flow charts and algorithms, that were used to produce these graphs and tables, accompany modern codes.

Several efforts on the computerized representation of codes in general, and structural codes in particular, are reported in the bibliography ([5]-[14], [20]-[22]). Pertinent investigations started in the late sixties by Professor Steven Fenves [5] and continue until today.

What is interesting today, is the development of integrated design systems that simulate the course of actions taken by experienced designers. The fundamental difference of current methods lies in their ability to use parts of the program on an "if needed" basis. With greater computational power becoming increasingly available, new methods and programming environments make this goal feasible. The knowledge about a domain can be incorporated into computer systems in the form of knowledge based expert systems (KBES) that, when properly structured, can manipulate a knowledge base using the inference engine to prove a list of goals and subgoals. All the forms of knowledge representation, used in AI applications, have been utilized to interpret design codes. Among these, the most popular are the production systems or rule based systems, the use of frames, which are being replaced by object oriented programming, and other forms of semantics such as decision-tables, information networks and organisation systems.

The decision tables, proposed by S. Fenves, as an extension to tabular decision logic, provide a systematic way to represent and process the requirements of design codes. Moreover, the data and the application rules can be logically channelled in the form of a network of decision tables. Special pre-processors have been developed to manipulate the decision tables and to incorporate them into analysis programs [11].

Information networks are a collection of nodes and branches, where each node represents a data item and each branch represents a relation between two nodes. The provisions of a code are represented in the form of a graph which results after parsing the provision. The leaves of the graph represent the basic data, while the items at higher levels represent evaluated data.

Various systems have also been proposed that correspond to various forms of Database Management Systems (DBMS), and aim at offering environments for multi-purpose access systems in integrated design. Among these, the object oriented DBMS systems seem to be the most popular [9], [21].

To render the "if needed" character of a code representation in a way that corresponds to the design of structures, appropriate design strategies must be employed. These may be either under the full control of the designer, leading to interactive design systems [14], or may be computerized design strategies that simulate the design process on the basis of decomposition, structural optimization, numerical experimentation etc., to attain a list of design goals. The development of ad-hoc design strategies, and heuristic or general design theories, presents considerable difficulties due to their synthetic nature. The existing design strategies tend to simulate the course of actions taken by designers, or address the problems of conceptual design and preliminary design of structures.

Moreover, the system must be in a position to reveal the appropriate list of conformance checks that are needed for the design of a particular component on the basis of its type, state of stress, its particular features and the type of analysis. In addition, hypertext system technologies may be used in

a productive way to enhance the usability of such systems, especially regarding the descriptive parts of a code [18].

In another direction, new tools and methods need to be developed to assist design standards writing organizations in checking the completeness, uniqueness, i.e. absence of redundancy and contradiction and correctness of the codes [7].

4. LOGICAL SCHEMES FOR THE REPRESENTATION AND PROCESSING OF DESIGN STANDARDS

The representation and processing of structural design codes can be performed also in a way that follows the intrinsic logic of the code, and transforms code clauses into formal logic using logic programming. This approach has been applied extensively in the form of rule based systems for descriptive codes, such as architectural codes [10], but its potential can be extended to cover also the algorithmic parts of codes [22], [23].

A logic program is a set of rules that define relations between data structures, while computation, in the context of logic programming, means to prove that a goal statement (rule) is true using the other defined rules. This process is constructive and provides the values for the goal variables, which constitute the results of the computation [15]. The implementation of logic programming in Prolog is performed by using a control mechanism based on sequential search with backtracking on a restricted class of logical theories, namely Horn clause theories. Prolog can be viewed also as a relational database programming language, which manipulates and alters the database information, that describes a particular domain.

The main advantage of a Prolog representation of the code requirements, is that the structure of the language is such that it sequentially demands the truth of the predicates of a rule. Generically, this enables the implementation of a sequence of requirements in one predicate by combining descriptive, i.e. declarative, and procedural predicates. Rules in Prolog may have several definitions, thus the appropriate rule is matched and processed on a "if needed" basis.

As a result, using Prolog as the means of expressing the design philosophy of the code, clarity and modularity are preserved as main characteristics. This implies that a separate module must exist that defines the partial safety factors for the actions and the material properties. Another module must define the loading cases and the load combinations that apply to different types of structures that correspond to the ultimate limit states. Yet another module must define all the resistances covered by the ultimate limit states of the code, together with the domain of their application. Finally, another module must control the serviceability limit state checks. These correspond to the standard modules that exist in every code. In addition, there may exist other modules that contain the particular information of the code that refers to the specific material, the detailing of the structural components, the connections, and the rules of good design practice that the code recommends. This information is tailor-made and corresponds to the particular structure of the specific code. These separate modules can be maintained easily by incorporating changes in the standards, and can be placed in parallel with the relevant parts of other codes in integrated systems that accommodate different design codes.

Feijo et. al. ([17],[18]), Jain, Law et. al. [22], and Rasdorf and Lakmazaheri[23] presented alternate models for the representation of design codes based on first-order logic and their incorporation into design automation systems. Feijo et. al. suggested also a formal link with hypertext systems, in which data is stored in a network of nodes connected by links. Rasdorf and Lakmazaheri put the emphasis on the axiomatic formulation of the organizational submodel and the representation and processing submodel of the code, which can be interrogated via theorem proving.

After building the separate modules, dominant part of the entire process becomes the control of the information flow. This is based on the sequential backtracking algorithm which is the searching mechanism of Prolog Language. Backtracking can be considered as a searching technique of all possible solutions in a systematic manner. Its description in procedural terms is as follows:

Having a predicate that depends on a list of n variables $[x_1,x_2,..,x_n]$ that participate in a particular check and are stated according to the safety format of the code. This list must satisfy a requirement or property $P_n[x_1,x_2,..,x_n]$. In general, we assume that x_i can take values from a set X_i. The assumption is, that all choices in a set X_i are linearly ordered. Once the values of $x_1,x_2,..,x_{k-1}$ are fixed, we select the smallest value x_k among the set X_k, which leads to a feasible list which satisfies $P_k[x_1,x_2,..,x_{k-1},x_k]$. The subset of X_k which represents feasible choices for x_k is denoted by S_k. Since X_k is ordered, all choices in S_k are also ordered. With these assumptions the general backtracking algorithm can be stated as follows [24], where S_k is formed simultaneously by checking the feasibility of x_k.

We start by examining S_k (k=1,2,...,n) sequentially.

Step 1. If S_k is not empty, set x_k to be the smallest value in S_k which has not been tried previously. If k<n, increase k by 1 and repeat the step. If k=n, record the list as a feasible list. If we want all feasible solutions, decrease k by 1 and repeat this step. Otherwise stop.

Step 2. If S_k is empty and k=1, no more feasible lists exist, and therefore stop. If S_k is empty and k>1, decrease k by 1 and return to step 1.

During the design process, following a backtracking algorithm, usually we associate a cost with a particular list which satisfies a relationship of the following form:

$$\text{cost of } [x_1,x_2,..,x_{k-1}] <= \text{cost of } [x_1,x_2,..,x_{k-1},x_k] \text{ for all } x_k \qquad (3)$$

During searching we do not branch from a node whose cost is higher than the minimum cost solution found so far. Of course, the bound is updated if a better solution is found. For a maximization problem we do exactly the opposite. We do not branch from a node whose value is less than the value of the maximum value solution. This corresponds to a branch and bound algorithm that usually prunes the search.

In processing code requirements, usually we have to satisfy a conjunction or disjunction of more than one demands on the structure. This can be accomplished using a sequential backtracking algorithm, which is based on the structure of a Horn clause, and has the following form:

$$A \rightarrow P_1, P_2, \ldots, P_n, \quad n > 0 \qquad (4)$$

The important feature of Horn clauses is that they can be read both declaratively, saying that A is true if P_1 and P_2 and... and P_n is true, and procedurally, saying that to solve problem A, one can solve subproblems P_1 and P_2 and ... and P_n. In order to establish whether A is true, a Prolog program attempts to prove that P_1 and P_2 ... and P_n are true, starting from left to right. It uses the internal unification routines of the language as a form of a bi-directional pattern matching. It assigns values to the variables of the rule P_1 in order to prove that it is true and subsequently moves to P_2 leaving a marker in between to remember that up to that point the truth of clause A has been established. Hence, the program, by invoking the internal unification routines which sequentially search for the right values of the variables, establishes the truth of P_1 and P_2 and ... P_n, and thus the truth of A, binding in the mean time the feasible list of design variables. All rules are of the form given in relation (4) where A is referred to as the head of the rule and the conjunction of P_1 and P_2 and ... P_n as the body of the rule. Every subrule P_i is a predicate or categorim which is either true or false.

In order to restrict backtracking and thus reduce unnecessary search, the predicate "cut", or symbolically "!", is used which in effect deletes the markers for backtracking up to that point and

thus works as a barrier that doesn't allow backtracking to previous subgoals. The predicate "fail" of Prolog is one that makes the rule to fail and thus causes backtracking.

Following the sequential backtracking algorithm, binding of the list of variables takes place and affects the subsequent search. This issue is very critical and must be handled with care during the development of the system, in relation with the structure of the databases used. As discussed before in the procedural description of the backtracking algorithm, the feasible sets must be ordered. Therefore, if the database is built in increasing order with respect of the design objective, and the user has selected a number of different types of sections, a number of alternative solutions will be deduced. The existing conflict can be resolved either on the basis of secondary criteria, i.e. list of preferences in profiles, or other technological constraints, or by retaining all the solutions.

The logic approach has been used in the design of steel roofs using Eurocode 3 [19], where also the analysis is based on a logical program. The conformance checks used for the ultimate limit states correspond to tension, compression, bending, and bending with shear checks.

The code requirements can always be posed in the form of Horn clauses as for example the compression requirements of Eurocode 3. Compression members are designed according to the requirements of paragraph 5.4.4 and 5.5.1 of Eurocode 3 for buckling resistance. The relevant part in Prolog is as follows:

```
compression_requirements(Type,Cnt,_,Min,Length,Iarea):-
        section(Type,Name,Height,Width,Thickness,Area,Weight,Jx,_),
        chosen_section(_,_,_,WeightOld),
        Cnt * Area >= Iarea,
        Cnt * Weight < WeightOld,
        Tmp = Jx / Area,
        Radius_of_gyration=sqrt(Tmp),
        buckling_curve_axis(Type,Height,Width,Thickness,Curve,_),
        imperfection_factor(Curve,Alfa),
        Lamda=Length/Radius_of_gyration,
        Lamda1=pi*(sqrt(modulus_of_elasticity/yield_stress) ),
        Lamda2=(Lamda/Lamda1)*sqrt(Alfa),
        F=0.5*(1+Alfa*(Lamda2-0.2)+Lamda2*Lamda2),
        X=1/(F+sqrt(F*F-Lamda2*Lamda2)),
        NcRd=Cnt * Area * yield_stress,
        NbRd=X*NcRd,
        abs(Min) <= NbRd,
        CntArea = Cnt * Area,
        CntWeight = Cnt * Weight,
        save_chosen_section(Type,Name,CntArea,CntWeight),fail.
compression_requirements(_,_,_,_,_,_):- !.
```

The names of the variables are chosen close to the names of the quantities in the code to facilitate understanding of the relations. The entire process works by interrogating the database "section" with the attributes of all the profiles. This relational database, that Prolog language accommodates, is used in the searching of a new section as compared to a previous one. After the comparison of a pair of sections, the process selects the best section and fails at the end to force searching of the entire database for the particular group of members. The descriptive part in this provision corresponds to the selection of a buckling curve according to the type of section, hollow, welded rolled, etc. and the thickness to height ratios listed in Table 5.6 of the code. Therefore, the above predicate does not only perform the relevant conformance check but is used to select the optimal section within a specific type of profile.

Thus, a conformance check based on the provisions of a code and the selection of the optimal section with respect of a design objective, are rather straightforward for a given member. The important issue though is the identification of the required list of conformance checks for a member or a list of members. This issue was first addressed by S. Fenves introducing the decision tables which can be implemented efficiently in Prolog. The main difficulty faced in this respect is the vague information contained in a code that rarely specifies the domain of application of particular checks especially under exceptional environmental conditions. The list of actions and relevant conformance checks that are referred to a specific component form the design list for the component. In this work this list is fixed, but can result from the consistency checks of the code as a separate task.

The conformance checks can be applied to an already analysed structure of given dimensions, layout member sizes and loads. The way of incorporating the code provisions to the design process is a more general problem and relates to the strategy used in altering the structural configuration. This can be addressed in the context of structural optimization methods incorporating the constraints imposed by the codes of practice. In a logical approach, these can be stated in the form of a sequence of demands imposed on the design. These demands can be expressed directly in the form of Horn clauses given in relation (4).

The area of the so-called technical code based structural optimization is seeking to define the optimal design of a structure with respect to a single or multi criterion objective, subject to the constraints imposed by the selected code. In this context, the identification of the list of active constraints-provisions that correspond to particular conformance checks has a particular importance. It can be used as an active set to evaluate the sensitivities of the design with respect of the design variables and thus, depending on the form of optimization problem, alter the lay-out, shape or sizing of the components of the structure. Moreover, for a number of iterations it can be used to limit the design process to local modifications and therefore accelerate the entire design process. Therefore, code requirements can be interpreted not only as barriers on the design, but also as boundaries reflecting important information about the response of the structure. The identification of the active constraints can be performed easily in Prolog.

The proposed general scheme has been used for the design of a roof of an industrial building or warehouse using plane trusses and continuous purlins [19].

Although the system described above can be transported to other applications developed within the same design philosophy, is not totally independent from the remaining modules of the entire system. A more formal representation scheme, as for example the one based on the object oriented paradigm presented by Garrett et. al. [21] can stand as an independent module. In that case, a quite cumbersome interface must be built to provide information needed for the optimization process. A similar situation exists in structural optimization programmes. Some of which use a separate module to calculate sensitivities and others use a combined approach. The differences are in the software development but affect also the computation time of the system, the combined one being faster.

5. SUMMARY AND CONCLUSIONS

A general scheme based on logic programming is employed in the representation of code provisions. The control of the flow of information, that is the dominant part of the conformance checks and the design procedure of a structure, is based on the sequential backtracking algorithm used in Prolog as a searching algorithm. The design space of a particular problem is formed as the product of the relational databases of the members of a structure and the constraints. Even though logic programming has been used extensively in code representation and processing the unified scheme presented addresses the overall problem within the structural optimization perspective of structural design.

REFERENCES

[1] EUROCODE NO 3: "Design of Steel Structures," Part 1, General Rules and Rules for Buildings, Final Draft, Issued to Liaison Engineers, February 1989.

[2] JOINT COMMITTEE ON STRUCTURAL SAFETY, CEB - CECM - CIB - FIP - IABSE - IASS - RILEM: "General Principles on Reliability for Structural Design", International Association for Bridge and Structural Engineering (IABSE), 1981.

[3] HWANG, H.H.M., USHIBA, H., and SHINOZUCA, M., "Reliability Analysis of Code-Designed Structures under Natural Hazards", Technical Report NCEER-88-0008, 1988.

[4] HWANG, H.H.M., and HSU, H-M., "A Study of Reliability-Based Criteria for Seismic Design of Reinforced Concrete Frame Buildings", NCEER-91-0023, 1991.

[5] FENVES, S.J., "Tabular Decision Logic for Structural Design", ASCE Journal of Structural Division, Vol. 92, No. ST6, December, pp. 473-490, 1966.

[6] HARRIS, J.R., and WRIGHT, R.N., "Organization of Building Standards: Systematic Techniques for Scope and Arrangement", Building Science Series NBS BSS 136, National Bureau of Standards, Washington, D.C., 1980.

[7] FENVES, S.J., and WRIGHT, R.N., "The Representation and Uses of Design Specifications", NBS Technical Note 940, 1977.

[8] GARRETT, JR.J.H., and FENVES, S.J., "Knowledge-Based Standard-Independent Member Design", ASCE J. of Structural Engineering, Vol. 115, No. 6, pp. 1396-1411, 1989.

[9] RASDORF, W.J., and WANG, T.E., "Generic Design Standards Processing in an Expert System Environment", ASCE Journal of Computing in Civil Engineering, Vol. 2, pp. 68-87, 1988.

[10] ROSENMAN, M.A, and GERO, J.S., "Design Codes as Expert Systems", Computer-Aided Design, 17(9), pp. 399-409, 1986.

[11] CRONEMBOLD, J.R., and LAW, K.H., "Automated Processing of Design Standards", ASCE Journal of Computing in Civil Engineering, Vol. 2, pp. 255-273, 1988.

[12] TOPPING, B.H.V., and KUMAR, B., "Knowledge Representation and Processing for Structural Engineering Design Codes", Engineering Applications of AI, Vol. 2, No. 3, pp. 214-228, 1989.

[13] BEDARD, C., and GOWRI, K., "Automatic Building Design Process with KBES", ASCE J. of Computing in Civil Engineering, Vol. 4, No. 2, April, pp. 69-83, 1990.

[14] TYSON, T.R., "Effective Automation for Structural Design", ASCE Journal of Computing in Civil Engineering, Vol. 5, No. 2, April, pp. 132-140, 1991.

[15] BRATKO, I., "PROLOG Programming for Artificial Intelligence", 2nd Edition, Addison-Wesley Publishing Co., 1990.

[16] STERLING, L. and SHAPIRO, E., "The Art of Prolog: Advanced Programming Techniques", MIT Press, Cambridge, Massachusetts, 1986.

[17] FEIJO, B., DOWLING, P.J., and SMITH, D.L., "Incorporation of Steel Design Codes into Design Automation Systems", Expert Systems in Civil Engineering, IABSE Colloquium, Bergamo, 1989.

[18] FEIJO, B., KRAUSE, W.G., SMITH, D.L., and DOWLING, P.J., "A Hypertext Model for Steel Design Codes", J. of Constructional Steel Research, Vol. 28, 167-186, 1994.

[19] KOUMOUSIS, V.K., and GEORGIOU, P.G, "An Expert System for Steel Roof Design", Structural Engineering Review, Vol. 5, No. 2, pp. 169-181, 1993.

[20] GARRETT, JR., J. H. and S.J. FENVES, "A Knowledge-Based Standard Processor for Structural Component Design", Engineering with Computers, 2(4) pp. 219-238, 1987.

[21] GARRETT, JR. H. and M. MAHER HAKIM, "An Object-Oriented Model of Engineering Design Standards", Journal of Computing in Civil Engineering, 6(3), pp. 323-347, 1992.

[22] JAIN D., K. H. LAW and H. KRAWINKLER, "On Processing Standards with Predicate Calculus", Proc. of the Sixth Conference on Computing in Civil Engineering, ASCE, Atlanta, GA, 1989.

[23] RASDORF, W.J. and S. LAKMAZAHERI, "Logic-Based Approach for Modelling Organization of Design Standards", ASCE Journal of Computing in Civil Engineering, 4(2), pp. 102-123, 1990.

[24] HU T. C. , "Combinatorial Algorithms", Addison-Wesley Pub., pp. 138-151, 1982.

[25] FENVES, S.,GARRETT, J.,HAKIM, M., "Representation and Processing of Design Standards: A Bifurcation between Research and Practice", 1994 ASCE Structures Congress, Atlanta, GA.

Leere Seite
Blank page
Page vide