Incorporation of Steel Design Codes into Design Automation Systems

Incorporation des normes de construction métallique dans des systèmes automatiques de projet

Einbeziehung der Normen für Stahlbau in automatische Projektsysteme

Bruno FEIJÓ
Aeronautics Engineer
PUC/RJ
Rio de Janeiro, Brazil

Patrick J. DOWLING
Professor of Civil Eng.
Imperial College
London, UK

David Lloyd SMITH
Civil Engineer
Imperial College
London, UK

Bruno Feijó received his Engineering degree at ITA, São Paulo, 1975 and his PhD at Imperial College, 1988. For many years he worked with computers in engineering design. He is now Lecturer of Computer Science in PUC/RJ. He is also Visiting Lecturer in the Expert Systems Lab, Imperial College.

Patrick Dowling is a member of Britain's National Academy of Engineering, the Fellowship of Engineering. He is Chairman of the Drafting Panel of Eurocode3 and Editor of the Journal of Construction Steel Research and Director of the Consulting firm Chapman & Dowling Associates Ltd which drafted the Brazilian Steel Bridge Code.

Dr David Smith is Head of the Systems and Mechanics Section of the Civil Engineering Department, Imperial College, London, and he is currently Director of Undergraduate Studies. He plays a leading role in the Civil Engineering Expert Systems Laboratory.

SUMMARY

This paper presents a model for the representation of design codes based on first-order logic and their incorporation into Design Automation Systems. Also, a formal link with hypertext systems is suggested. Furthermore, presented are the results of the initial investigation into the logical structure of the British Code BS5950 for steel design.

RESUME

Cet article présente un modèle pour la représentation des normes de projet basé sur une logique de premier ordre et leur introduction dans des Systèmes Automatiques de Projet. Une liaison formelle avec un système de type hypertexte est également présentée. Les résultats d'une investigation préliminaire sur la norme anglaise BS5950 concernant les structures métalliques sont également présentés.

ZUSAMMENFASSUNG

Dieser Artikel stellt ein Modell für die Beschreibung von auf logischen Grundregeln aufbauenden Projektnormen und deren Einbeziehung in automatische Projektsysteme vor. Eine formale Verbindung mit Hypertext Systemen wird auch angeregt. Die Ergebnisse einer ersten Analyse der logischen Struktur des British Code BS5950 für Stahlbauprojekte werden präsentiert.

1. INTRODUCTION

This paper presents a model for the representation of design codes that can be easily incorporated into Design Automation Systems. This model is based on and-or graphs and first-order logic. Furthermore, this model suggests a formal link between design codes and hypertext systems - an emerging field in engineering information management. Also, presented are the results of the initial investigation into the logical structure of the British Code BS5950 for steel design. The practical experience of the authors with the present model is restricted to the domain of steel design. However, the model may be applied to any design code.

2. THE PROBLEM OF KNOWLEDGE REPRESENTATION

The study of regulations as a type of knowledge can be found in the areas of Office Automation [1] and Legal Reasoning [2][3]. In Design Research, this type of knowledge presents the following characteristics:

- it is supposed to have an explicit and precise interpretation (in contrast with the "open nature" of law in the field of legal reasoning);
- it is available in written form;
- it is supposed to be complete and correct;
- it presents no uncertain facts;
- it requires no vague reasoning;
- it presents a simple structure of discourse (in contrast with the complex structures found in the area of Text Generation [4]).

The representation of this knowledge in an artificially intelligent system involves two main tasks: its structuring and the translation of the written provisions into a form suitable for symbolic manipulation. However, these tasks are not easily accomplished, because they should be carried out with the entire design process in mind. Moreover, revisions of the text are required when the knowledge is incomplete and/or incorrect[a]. In this case, a knowledge engineer might be required to reveal the results intended by the code writers.

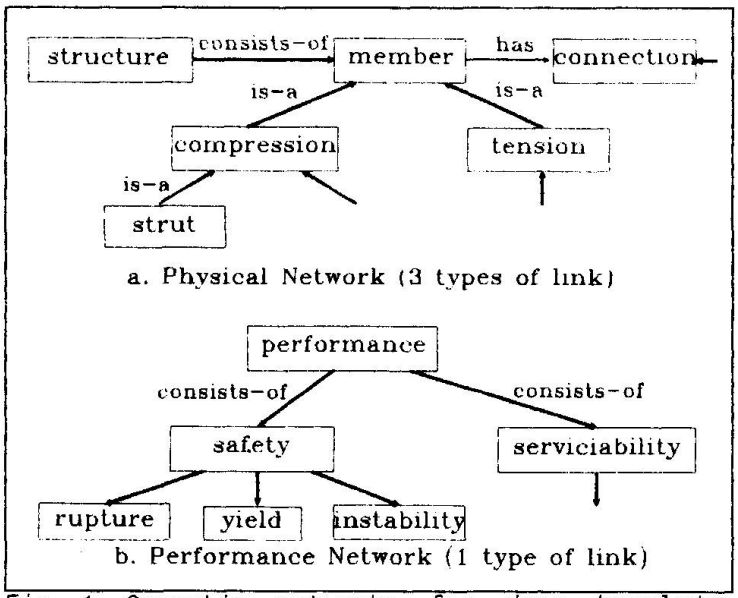

Fig. 1 Semantic networks for deep knowledge representation

Any type of knowledge representation in design should use deep models (i.e. models with a causal structure underlying the more external structure). As far as design codes are concerned, any deep model will certainly be biased by the work of Fenves, Wright and Harris [5][6]. In this

context, one could produce semantic networks like those in Fig.1. This paper assumes the existence of such deep structures and focusses on more external representations of a design code. The proposed representation is based on and-or graphs and first-order logic.

3. THE GRAPH REPRESENTATION

3.1 The design code graph

The structure of a design code can be represented by an and-or graph[b] in which provisions call subprovisions (Fig.2). Furthermore, attributes A_i are attached to each node i. These attributes contain additional information about the provisions, such as the Limit States (LS) governing the provision, the number of the section, a copy of the original text and one or more labels for classification. For example:

A_3 =
{ LS(yielding, rupture), 4.6, "For tension members with...", ... }

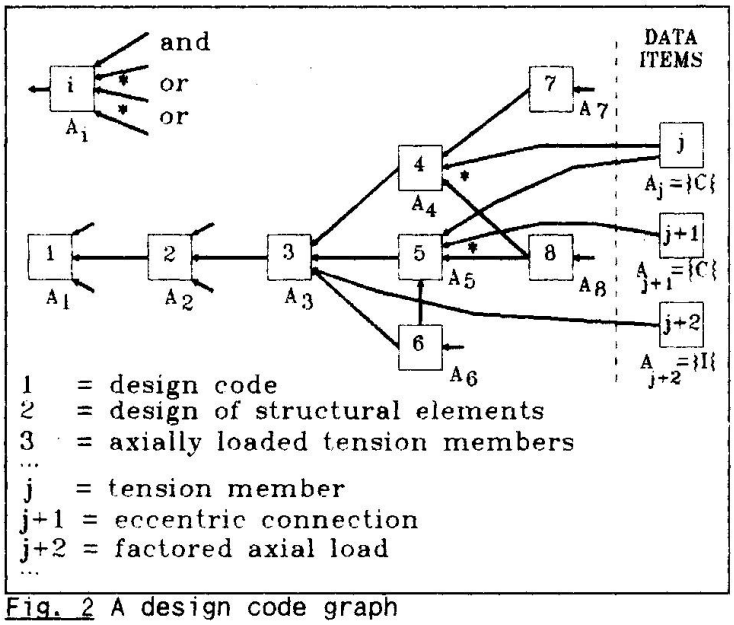

Fig. 2 A design code graph

Data items must be added to the and-or graph as terminal nodes. A data item should be identified by its type (which is an attribute) according to the following classification:

I INPUT. For example: "factored axial load";

C CLASSIFIER. It is a special kind of input that classifies an entity and makes a provision entitled for conformance checking (i.e. the process of checking a design entity for conformance with a code). For example: "tension member";

D DEFINITION. It is a data item used in definitions. For example: "the area of a hole is calculated in the place of its axis" is used in the definition of "hole area";

E EXTERNAL. It is a data item that is given after an external procedure is executed. Sometimes the external procedure is a call to the design code itself. For example: checking a double angle requires that "the slenderness of each component does not exceed 80" (in turn, this may require a consultation of the code).

The and-or graph with attached attributes and data items is called a design code graph.

3.2 Properties

Design code graphs present the following properties:

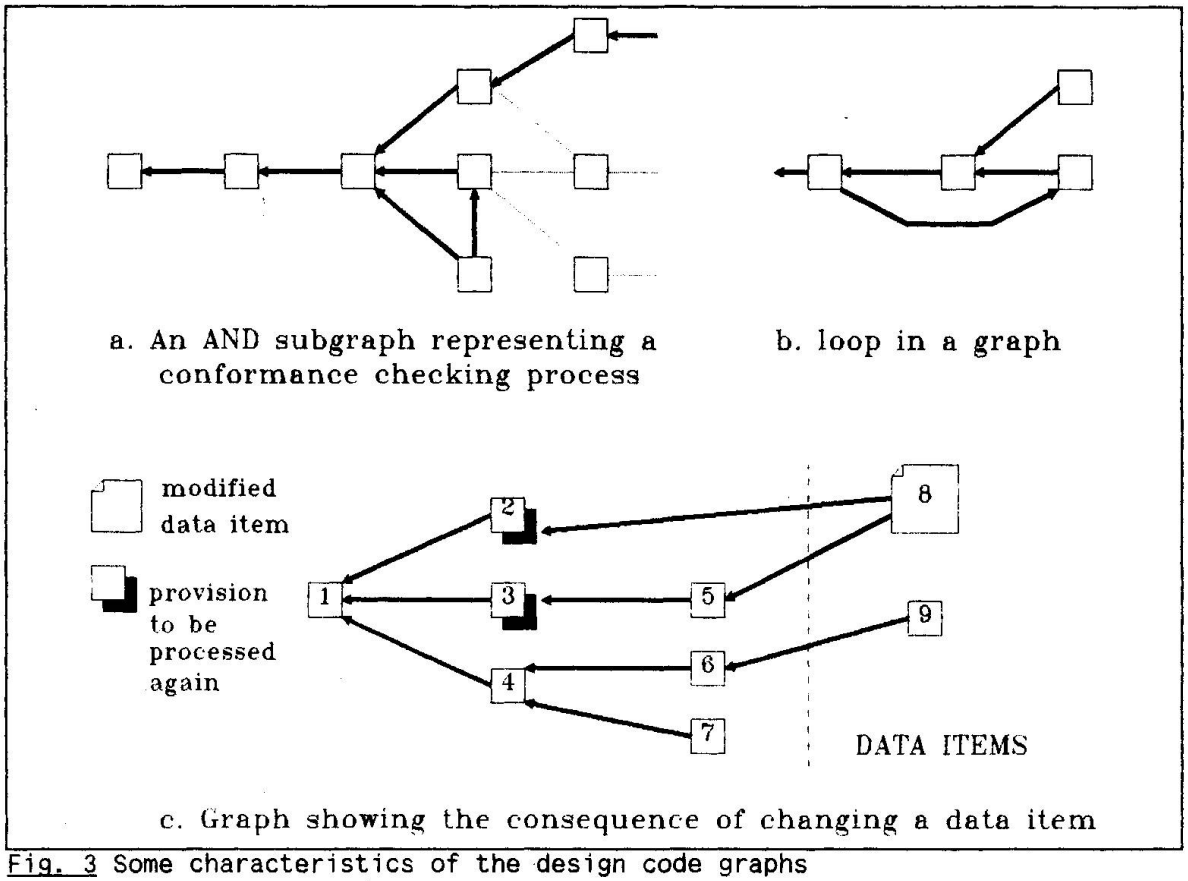

Fig. 3 Some characteristics of the design code graphs

1. Conformance checking should be represented by a unique AND subgraph, otherwise the code presents ambiguities or multiple interpretations (Fig. 3.a).

2. Cumbersome forms of cross-referencing, lack of connections and loops (Fig. 3.b) can be easily detected with the help of the design code graphs.

3. The graphs (and especially the subgraphs) show the consequences of changing a particular data item (Fig. 3.c). This property can be used to support "intelligent decisions" made by Design Automation Systems. Also, it allows an effective redesign.

4. Design code graphs share some properties with the information networks proposed by Fenves and Wright [5]. In particular, the graph can easily be converted into a depth-first spanning tree [7]. This tree can be used by writers of design codes to identify cross references and to achieve better textual expression.

4. THE HORN CLAUSE SYSTEM

The design code graph of Figure 2 does not represent a design code in its entirety, albeit it represents its overall organization. The representation is completed by attaching a set of Horn clauses H_i to each node i. These Horn clauses must be mutually exclusive rules which should define the provisions completely (Fig.4). Moreover, the set H_i can contain conditions which are not presented in the and-or graph, such as equations. The mutually exclusive rules assure the property of uniqueness (i.e. that the design code yields one and only one result).

The main variable X in the set of Horn clauses represents an entity to be designed, such as a member or an element of a member. Only one kind of entity

should be associated with the design code each time the code is invoked. This simple semantics yields a more robust model.

The union of the sets H_i forms a logic program P, called a design code program, i.e.

$$P = H_1 \cup H_2 \cup \ldots \cup H_n$$

The program P (or any subset of it) should be invoked by a query that extracts only one answer. This restriction is required by the property of uniqueness.

The most important characteristic of P is that it contains the structure of the graph that uses P to be defined. This recursive aspect of the model yields a simple and concise conclusion:

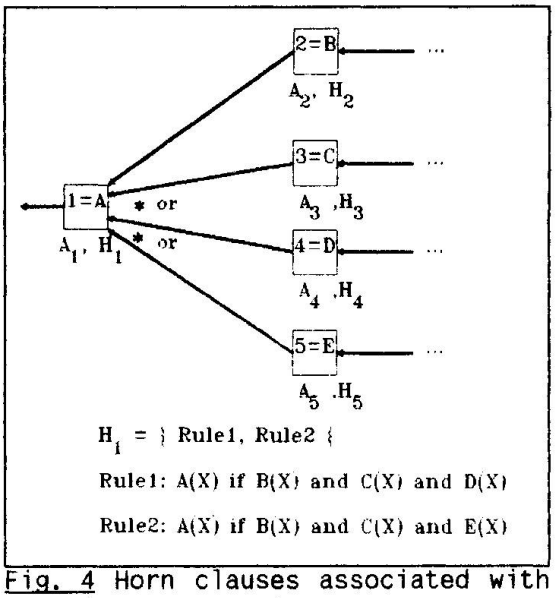

Fig. 4 Horn clauses associated with a design code graph

"a design code can be entirely represented by the program P if the attributes A_i are attached to each member of H_i".

For example, the provision A of Fig.4 might be represented by:

A(X) if B(X) and C(X) and D(X) ; A_1
A(X) if B(X) and C(X) and E(X) ; A_1

In this approach, each rule contains two kinds of information: one concerning logic (inherent to any logic program) and the other concerning the structure and organization of the code (the design code graph).

5. A RESTRICTED FORM OF HYPERTEXT

The design code program presented in Section 4 can be understood as a restricted form of hypertext[c]. In this case, nodes are predicates, links are represented by the logic program P and node bodies are the attributes A_i. Naturally, the properties of design code graphs are valid for this form of hypertext which is called a design code hypertext. A prototype of this hypertext system has not yet been implemented by the authors.

Some of the principles underlying the system KMS [8] seem to be very appropriate to design code hypertexts. Hence, nodes could be displayed as windows with four components (Fig.5): Node Header, Node Body, Logical Items and Command Items.

In the design code hypertext there is just one type of link (although a second type similar to KMS Annotation Items could be considered). Hence, a link is not an object but a property of an item. The process of editing nodes is supposed to be similar to that found in KMS.

Every aspect of the design code hypertext has a counterpart in the logic program P and vice-versa. For example, querying P is like an operation of automatic navigation. Moreover, if the querying mode is interactive (i.e. the system stops and asks for missing facts), then conventional navigation might be available temporarily. Also proof trees might be available for conventional navigation. There are many other aspects to be explored, such as: Selection by freezing

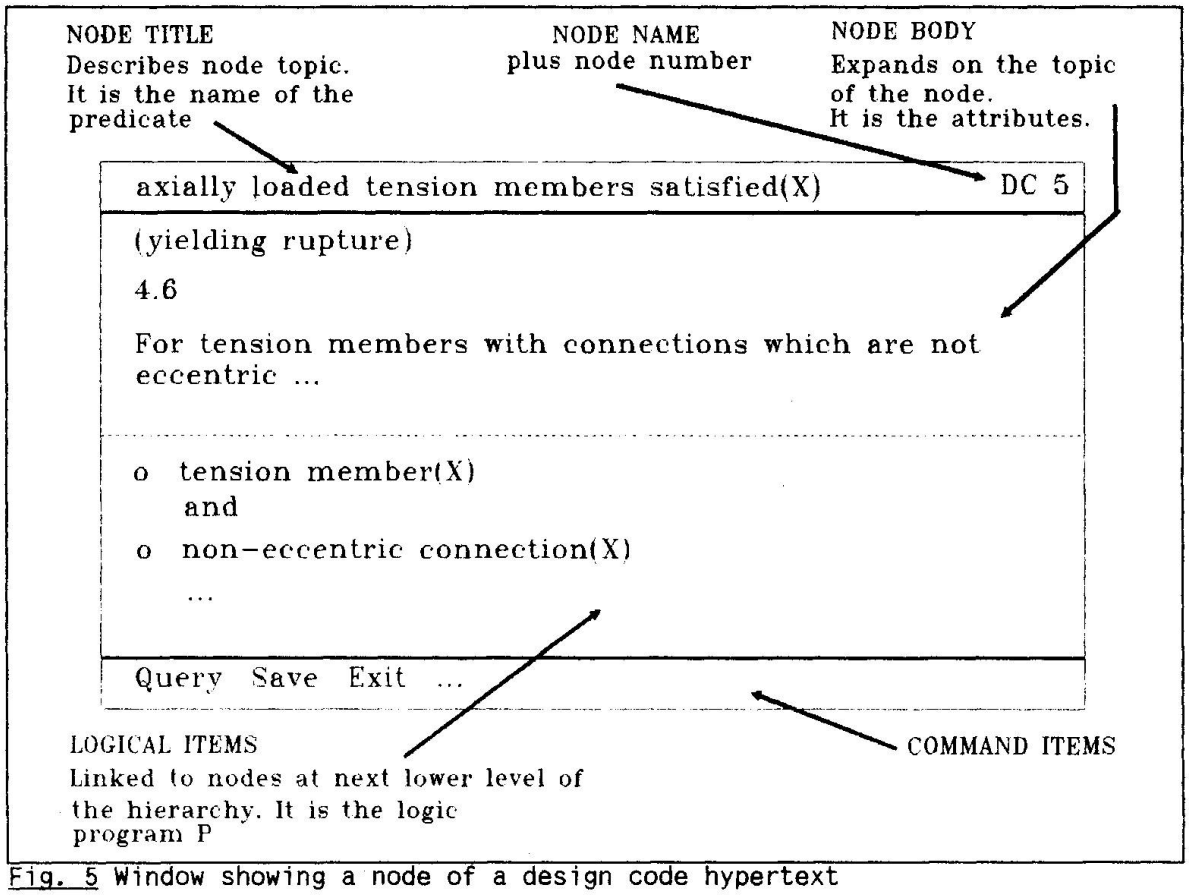

Fig. 5 Window showing a node of a design code hypertext

undesirable provisions (e.g. those concerning serviceability as limit states); Priority for searching (e.g. yielding provisions first); Consequences of changing data items; Dynamic incorporation of provisions, e.g. A(X) if B(X) and assert(A(X)); Explanation; Analogy based on past episodes; Debugging. For instance, a unique metaknowledge can be used to guide both the logical inference and the hypertext navigation (by freezing part of the program/network).

6. DESIGN CODES AND DESIGN AUTOMATION SYSTEMS

Design codes can be effectively incorporated into Design Automation Systems if their models have the same mathematical formalism. This common framework, the authors believe, should be first-order logic.

The model SAE [9] is based on first-order logic and considers design as a recursive process involving Synthesis, Analysis and Evaluation. In this model, the current state of design is represented by a set of facts in predicate form, e.g. "tension-member(b1)". These facts build up the Database of Design Facts (DDF) and are in canonical form (i.e. the standard form used by a specific design code). The translation of a fact from a general form into a canonical form should be done at a higher level process of the model SAE.

The facts in DDF are arranged in three areas that are interpreted by the design code model as follows:

- **DESIGN REQUIREMENTS.** These facts represent a design request which includes hard constraints and basic assumptions. For example: "tension-member(beam1)";

- **DESIGN OPTIONS.** These facts are soft constraints from a design script. For example: "lacing-system(m1)";

- **DATA.** These facts are temporary data retrieved from the conventional database or generated by a subprocess. For example: "area(beam1, 0.5)" from the database; "tension-capacity(beam1, 235)" deduced within a provision; "axially loaded tension member satisfied(beam1)" as a provision that was satisfied.

Synthesis processes may decide to move a fact from an area to another area at any time.

The design code model proposed in this paper can be used by the model SAE in several ways. First, a higher level process (e.g. Preliminary Design) can use the design code as a logic program for determining appropriate design requirements. These requirements (usually related to limit states) are then used as hard constraints (i.e. performance specifications). For example: maximum thickness, load factors and maximum deflection.

Second, conformance checking can be invoked at any time as an analysis process, what is usually done in a lower level (e.g. Detailing). In this case, the query to the logic program can invoke the entire code, e.g.

BS5950_satisfied(beam1) ?

or part of it, e.g.

axially_loaded_tension_member_satisfied(beam1) ?

If a query fails, the reason for failure (a fact) should be available for some sort of intelligente decision making.

Third, the logic program can use a built-in predicate (e.g. assert) to add provisions to DDF if they hold, e.g. A(X) if B(X) and assert(A(X)). This technique saves processing time if those provisions are called again.

Forth, meta-knowledge can be exercised in the present model by using the attributes A_i. Finally, tests of inconsistency and redundancy during the process of using the design code (see Chapter 7) can be easily implemented. A preliminary version of a design code based on the present model was written in C by the authors.

7. INCONSISTENCY AND REDUNDANCY

The user or any Artificial Design Assistant of the model SAE may introduce inconsistencies into the knowledge database at any time. For example, he/she/it may state that the member is a "tension member" and later state that the member is a "strut" (i.e. a member of a structure carrying predominantly compressive axial load), which is a contradiction. There is also the problem of redundancy. For example, he/she/it may state that "the grade of steel is 43A" and "the flange thickness is 16 mm", and may state later that "design strength is 275 N/mm^2". This last sentence is a redundancy[d], because design strength is a logical consequence of the previous data.

First-order logic allows one to look for classical inconsistencies introduced into the database each time a new fact is given. For example, if "strut(beam1)" is given to the following knowledge database:

1. if strud(X) then compression_member(X)

2. not(tension_member(X) and compression_member(X))
3. tension_member(beam1)

it will create an inconsistent knowledge because both "strut(beam1)" and "not strut(beam1)" are true.

The procedure for redundancy tests is the following: a new fact should not be added to the database if it could be obtained as a consequence of the logical system. In very large and complex knowledge databases, it should be more practical to reject facts that can be proved in less than a specific number of steps (say 3 or 4).

8. THE DESIGN CODE BS5950

An investigation into the structure of the code BS5950 [10] has revealed some problems. First, the design code graph of the current version of the code presents a cumbersome structure due to an excess of cross referencing, as shown in Fig.6.

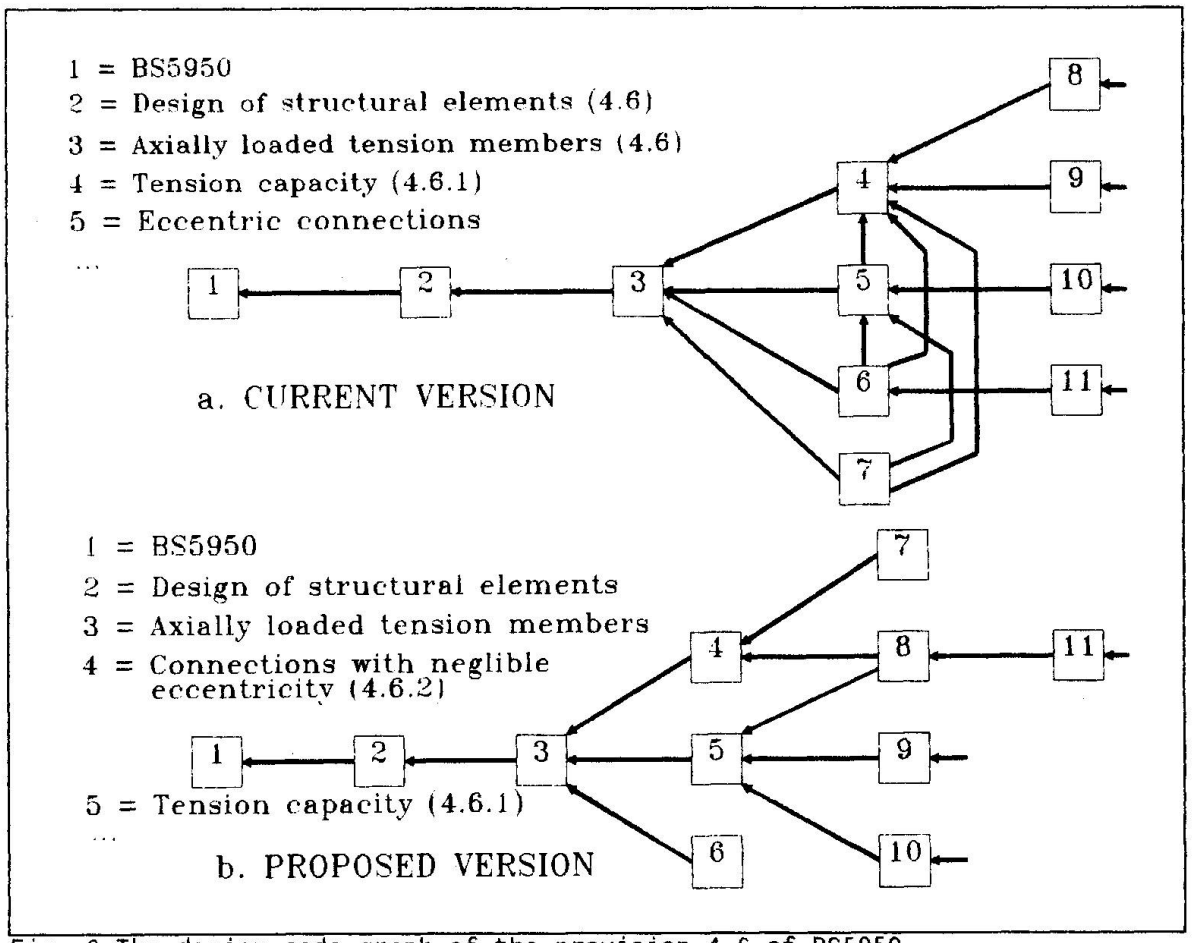

Fig. 6 The design code graph of the provision 4.6 of BS5950

Second, some provisions provide no general guidance at the first entry level of the provision (e.g. provision 4.6).

Third, the knowledge embedded in some provisions is not clearly expressed in the text. For example, the actions to be taken in provision 4.6 are not explicitly related to the three exclusive conditions about connections: (1) not eccentric; (2) eccentric; (3) eccentric, but the effect of the moments may be negleted.

Forth, the original text contains the objectionable word "except". This word has a metalevel interpretation in logic ("expect A" means "if A cannot be proved") that complicates deductions for humans. The semantics of "except" is a special case of negation called "negation as failure". Although this type of negation solves many problems it is advisable to avoid it (specially if nested negations may occur). This word could be easily removed from a design code by changing the structure of some provisions.

Fifth, some provisions permit undesirable logical conclusions. For example, from the point of view of methematical logic, the section 3.4.3 allows one to design any configuration of staggered holes. However, this is not the intention of the code writers. Irregular lay-outs of holes should not be permitted because the provision made in 3.4.3 is based upon empirical conclusions for regular distributions of holes.

9. CONCLUSIONS

This paper shows that logic leads to a simple, compact and robust representational model of a design code. Furthermore, logic can be used as a common framework for the design code model and the design process model. In this approach, inconsistency is gracefully handled by using classical negation in logic. Moreover, redundancy tests are easily implemented by a deduction mechanism.

The proposed design code model is simpler and, in some aspects, more robust than those proposed by other authors [11][12][13]. However, many aspects of those models can coexist with the present model. For instance, decision tables can be used by the knowledge engineer as a support tool during the building of a design code program.

Some problems of specific design codes may be revealed during the building of a design code program. For example, a partial analysis of BS5950 revealed cumbersome forms of cross referencing, hidden knowledge and texts that permit undesirable logical conclusions. These distortions in the structure of a code cause no harm to experienced engineers. However, Design Automation Systems require a clear and formal representation of the design code.

There are many ways in which the present work might be extended, such as: incorporating the design code model into systems with deep knowledge; further investigation into the hypertext nature of the model; a complete analysis of a code and its issue as a logic program or a hypertext system (as an alternative to its textual version); the study of problems associated with very large knowledge databases; further analysis of design codes in the light of the methods found in the field of Text Generation and Understanding.

NOTES

a) Incorrect in the sense that the results are not those intended by the code writers.

b) By definition, and-or graphs assume the inclusive interpretation of "OR", i.e. at least one (but possibly more) subprovisions hold.

c) Hypertext (or generally speaking: hypermedia) is a form of eletronic document in which data is stored in a network of nodes connect by links. Nodes can contain text, graphics, sound, programs to be executed or other forms of data. The entire network and individual nodes are displayed through an window system.

Users navigate in a hypertext database by pointing the mouse cursor at an item which has a mark to indicate a link to another node.

d) It could be an inconsistency if a contradictory value is given, e.g. "design strength is 355 N/mm^2".

ACKNOWLEDGEMENTS

The authors would like to thank the Steel Construction Institute for the financial assistance.

REFERENCES

1. PAOLINI,P. et al., Knowledge based document generation. In Lamersdorf, W.(ed.), Office Knowledge: Representation, Management, and Utilization, Elsevier Science Publ., North-Holland, IFIP, 1988, p.179-195.

2. SERGOT,M., KOWALSKI,R.A., KRIWACZEK,P. HAMMOND,P. and CORY,H.T., The British Nationality Act as a Logic Program, Commun. ACM, 29, (5), May 1986.

3. PINHEIRO,C.S. and SCHWABE,D., Expert systems and social welfare benefits regulations: the Brazilian Case, Comp. Science Dept, PUCRio, Brazil, 1988.

4. MANN,W.C., Text generation: the problem of text structure, in McDonald,D. and Bolc,L. (eds.), Natural Language Generation Systems, Springer-Verlag, 1988, p. 47-68.

5. FENVES,S.J. and WRIGHT,R.N., The representation and use of design specifications, in Hall,W.J. (ed.), Structural and Geotechnical Mechanics, Prentice-Hall, 1977, p. 278-304.

6. HARRIS,J.R. and WRIGHT,R.N., Organization of building standards: systematic techniques for scope and arrangement, NBS Building Science Series 136, 1981.

7. AHO,A.V., HOPCROFT,J.E. and ULLMAN,J.D., The Design and Analysis of Computer Algorithms, Addison-Wesley, 1974.

8. AKSCYN,R.M. et al., KMS: a distributed hypermedia system for managing knowledge in organizations, Commun. ACM, 31, (7), July 1988, p.820-835.

9. BENTO,J, FEIJO,B. and DOWLING,P.J., The knowledge based design of steel portal frames for agricultural buildings, IABSE Colloquium, Expert Systems in Civil Engineering (to appear).

10. BRITISH STANDARDS INSTITUTION, BS5950: Structural Use of Steelwork in Buildings, Part 1, England, 1985.

12. ROSENMAN,M.A., GERO,J.S. and OXMAN,R., An expert system for design codes and design rules, in Sriram,D. and Adey,R. (eds.), Applications of Artificial Intelligence to Engineering Problems, Springer-Verlag, 1986, p.745-758.

13. THOMSON,J. et al., Extending Prolog to provide better support for design code expert systems, Microcomputers in Civil Engineering, 3, 1988, p.93-109.