15. L.M. Chounet, R. Fauconnier, J. Sornay, "Annexe Technique au Projet de Convention du Groupement d'Etudes et Recherches ALMETH, AFME, Paris, septembre (1984)

16. J.Hirsch, E. Sowell, J.A. Clarke, "A Proposal to Develop a Kernel System for the NextGeneration of Building Energy Simulation Software", BESG Group, LBL, Berkeley, november (1985)

17. ASHRAE, "Procedure for Determining Heating and Cooling Loads for Computerizing Energy Calculations - Algorithms for Building Heat Transfer Subroutines", ASHRAE Task Group, Atlanta, february (1975)

18. J. Lebrun, "Exercices de Simulation Réalisés dans le Cadre de l'Agence Internationale de l'Energie", Extrait du Rapport Final de la Recherche, LPB, Université de Liège, Liège, (1983)

19. J.A. Clarke, L. Laret, "Explanation of the Data Processor Proforma", ABACUS (Glasgow) and Laboratoire de Physique du Bâtiment (Liège), Working Document, december (1984)

20. GER ALMETH, "Description Normalisée (PROFORMA) des Modèles de Comportement Thermique de Bâtiment", Compte rendu du Séminaire ALMETH Proforma, Lyon, 7-8 novembre (1985)

21. B. Delcambre, "Vers une C.A.O. Intégrée des Projets de Bâtiments", Proc. of the 10th International CIB Congress on Advancing Building Technology, Washington, D.C., september (1986)

Formulation of Building Regulations using Interactive Logic Programs

David Stone and David A. Wilcox

Building Directorate
Scottish Development Department
Scottish Office
Edinburgh EH1 3SZ, U.K.

KEYWORDS

Building Regulations, Expert Systems, Logic Programs, PROLOG

ABSTRACT

The Directorate is responsible for the Building Standards (Scotland) Regulations which set out statutory technical requirements for building design and construction. As a part of a major review programme the Directorate is developing the use of interactive logic-based programs to assist in the formulation, evaluation and drafting of revised Regulations. The Regulations are formalized as a set of rules and definitions in logic programs written in micro-PROLOG. Such programs have both a declarative and a procedural interpretation. Declaratively the program can be regarded as a specification of the required Regulation which can be progressively developed and refined in a top-down manner. The formalization clarifies the logical structure of the Regulation and helps to avoid syntactic or semantic ambiguities. Procedurally the formalization can be run as an interactive program and the logical consequences of applying a set of Regulation requirements in any context examined. A trace of the computation records the network of rules and definitions used to arrive at a given conclusion. The inherent modularity of logic programs allows changes and modifications to be easily introduced and their effects tested. Building Regulations developed and formalized as logic programs constitute a ready-made knowledge base for use in expert systems applications.

Formulation des Règlements Se Rapportant Au Bâtiment, Utilisant des Programmes Logiques Interactifs

David Stone and David A. Wilcox

Building Directorate
Scottish Development Department
Scottish Office
Edinburgh EH1 3SZ, U.K.

MOTS-CLES

Programmes Logique, PROLOG, Reglements se Rapportant au Bâtiment, Systèmes d'Application Experts.

SOMMAIRE

Le conseil d'administration est responsable des règlements se rapportant au bâtiment (Ecosse) qui spécifient les préscriptions techniques règlementaries pour le dessein et la construction des bâtiments. Au cours d'un examen important de ses fonctions, le conseil d'administration poursuit le développement de l'utilisation des programmes logiques interactifs qui serviront à formuler, à évaluer, et à rédiger les règlements révisés. Les règlements sont formalisés en série de règles et définitions qui font parti des programmes logiques écrits en micro-PROLOG. De tels programmes ont à la fois une interprétation déclarative et de processus. Déclarativement, les programmes peuvent être considérés comme une spécification de la règlementation nécessaire qui peut être développée progressivement et améliorée de haut en bas. La formalisation éclaircit la structure logique de la règlementation et contribue à éviter des ambiguités syntactiques et sémantiques. Selon le processus, la formalisation peut opérer comme un programme interactif et les conséquences logiques d'appliquer un groupe de règlementations exigées quel que soit le contexte examiné. Un tracé d'un calcul enregistre le raison des règles et définitions utilisés pour en arriver à une conclusion donnée. La modularité inhérente des programmes logiques permet l'introduction facile de changements et modifications et l'examen de leurs effets. Les règlements se rapportant au bâtiment développés et formalisés en programmes logiques constituent une base de connaissances toutes faites à utiliser dans des systèmes d'application experts.

1.0 Introduction

The Building Directorate of the Scottish Development Department is responsible for the Building Standards (Scotland) Regulations. The Regulations are intended to safeguard public health and safety in the built environment and set out statutory technical requirements for building design and construction . The Directorate is currently engaged in a major review programme which is intended to revise and simplify both the format of the Regulations and their technical content. As a part of this review programme the Directorate has been developing the use of computer-based analytic methods to assist in the revision process.

The problems inherent in expressing and organizing complex rules and regulations in text form are well documented (1,2,3). The absence of recognised formal methods for the specification and testing of proposed regulations often results, despite the best intentions of their authors, in regulations with in-built inconsistencies which can lead to differing interpretations by users and enforcing authorities. In addition, the inappropriate organization of regulation texts often impedes easy reference to requirements and effectively obscures their intention.

Models of the information structures inherent in regulations and codes have been proposed as a basis for the development of systematic methods for the formulation, evaluation and drafting of regulations material (4,5,6). Fenves et al.(6,7) proposed a three level model of regulatory information using decision logic tables to model the content of individual requirements, information networks to model the relationships between items of information and argument trees to model the intent and scope of a set of provisions. Fenves (ibid) showed how the practical advantage of these methods could extend to all aspects of regulations work, from the initial collation of the information content of provisions, to the formal analysis of provisions to ensure certain important syntactic properties and finally to the expression and ordering of provisions in text form. In addition, the information model and analytic methods proposed could be implemented in computer-based information systems. Stahl(8) and Gero(9) report on implementations of the Fenves model in computer systems for building code writers.

The current work of the Directorate is similarly concerned with analytic methods for the specification and testing of regulations and their implementation in computer systems and in particular in computer systems using interactive logic programs. Logic programming provides a direct and natural way of implementing the logical features of the kind of information model outlined above and, at the same time, offers a number of advantages over conventional software systems.

2.0 Logic programming

The objective of logic programming is to provide a means of defining computer applications in terms of a machine intelligible form of symbolic logic . Symbolic logic is based on propositional logic which is concerned with expressing propositions and the relationships between propositions and determining how one proposition can be validly inferred from another; a proposition in this sense is simply a statement which may be either true or

false. What uniquely distinguishes logic programming from conventional programming is that by using logic as a descriptive formalism and logical inference as a computational mechanism, logic programming effectively separates the description of a problem from the computational behaviour necessary to solve it.

The most practical and efficient implementation of the notion of logic programming at the moment is the programming language PROLOG and it is a version of this language, micro-PROLOG, which is in use in the work reported here. All program statements in PROLOG correspond to a restricted form of sentence called clauses. Clauses are implications of the form:-

(this conclusion is true) IF (this set of conditions is true)

or, alternatively:-

A if B and C and ..... and D

A logic program consists of a set of such clauses which represent either facts or rules about the domain of interest. A fact is merely a conclusion with no conditions. For example:-

warehouse occupant-load-factor 27.9

is a fact whilst:-

```
X total-width escape-route if
              Y calculated-total-width escape-route and
              Z minimum-total-width escape-route and
              X greater-of (Y Z)
```

is a rule. It is this combination of rules and facts, expressed in clausal form, which enables the representation in a computer system of the information contained in regulation requirements.

All computation in logic programming is a process of logical inference, that is the process of establishing what can be logically deduced from a set of rules and facts in response to some query. This inferencing mechanism is an in-built property of PROLOG and, in effect, is what makes it a programming language. Whereas in a conventional programming language there is a clear distinction between program and data, in logic programming this distinction is blurred; a set of logic program clauses can be regarded as both data and program. That is to say, a set of clauses has both a declarative and a procedural interpretation (10). For example, for the rule shown above a declarative reading is:-

"the total width for an escape route is the greater of
the total width calculated as necessary and the minimum
total width allowed"

whilst a procedural interpretation is:-

"to derive the total width of an escape route, calculate
the total width required, find the minimum total width
allowed and take the greater of the two"

A set of logic program clauses, then, can be regarded as both database and program with logic as a shared formalism and logical inference as the shared model of information retrieval. Most of the advantages of logic programming as an environment for the specification and testing of regulations derive from this dual interpretation of logic program clauses as both description and process.

2.1 Logic programs as specification

The formalization of regulation requirements as a logic program data-base has a number of benefits from the declarative or descriptive point of view. Regulations, by their very nature, map quite readily into the rule and fact clausal form. This process of formalization itself can often help clarify the logic of a requirement. The consequent database of rules and facts represents a concise and logical description of the regulation requirements which, even in the precise syntax of PROLOG, remain expressive and easily read. The inherently modular structure of the clausal form is itself of benefit. Each clause isolates a fragment of information and it is the accumulation of such fragments which enables the construction of more complex requirements. This modularity encourages an incremental, top-down development of regulation requirements. High level rules can be defined without the rules or facts governing their conditions necessarily having been defined. During development, information can be added, deleted or modified easily, either to clarify a requirement or to test an alternative formulation. This combination of expressiveness and modularity enables the rule database to be regarded as a specification of a regulation which can be progressively tested and refined during the formulative stages of defining regulation requirements.

2.2 Empirical and formal tests of program properties

Fenves (7) identified three important properties which regulatory information should exhibit in use. These are:-

1) regulation requirements should be complete: that is to say for any possible set of values to rule conditions the database must yield an explicit requirement or conclusion.

2) regulation requirements should be unique: that is to say for any given set of values to rule conditions the database should yield only one requirement or conclusion.

3) A set of regulations should be correct: that is to say the regulations should yield the result intended by their author.

The first two of these properties are concerned with the syntax or structure of the rule database whilst the third is concerned with the semantics or meaning of the rules. A procedural view of the database, that is the fact and rule clauses interpreted as a program , enables the specification of a regulation to be empirically tested for these properties. In order to describe this it is necessary to elaborate a little on PROLOG's inferencing mechanism and clausal structure.

All computation in logic programming is initiated by addressing a query to the rule database. PROLOG attempts to respond by matching the query pattern to a fact or rule conclusion in the database. When a match is found, rule conditions then represent goals to be proved. Rule conditions can themselves be defined by other rules or facts in the database. For example, in the rule shown earlier the conditions, "calculated-total-width" and "minimum-total-width", are defined by other rules thus:-

```
X calculated-total-width escape-route if
            Y occupant-capacity and
            TIMES(Y 5.3 X)
X minimum-total-width escape-route if
            Z minimum-width escape-route and
            x number-of-routes and
            TIMES(x Z X)
```

The condition for the minimum width of an escape route in the second rule is itself defined by two further rules. This kind of interdependant, hierarchical structure has been identified as characteristic of regulatory information and the basis of much of its complexity (6). The rule-based, clausal form of logic programming naturally implements such hierarchical structures and PROLOG's built-in inferencing strategy, a top-down, recursive search through a tree of goals, enables the logical consequences of such structures to be examined.

Not all rule conditions, however, will be defined in the database. Some will depend upon circumstantial data, that is, upon particular design proposals and configurations. In these circumstances, the behaviour of PROLOG's inferencing mechanism must be enhanced to include the assumption that values for rule conditions not defined in the database can be solicited from the user. The empirical testing of a rule database for the properties outlined above is, then, an interactive process with the user querying the database and the system, where required, soliciting data values from the user. In this way, the database can be tested for completeness by ensuring that, for various sets of condition values, the system does not fail to generate a requirement. Similarly, the database can be tested for uniqueness by exploiting the recursive nature of the inferencing mechanism. After the confirmation of a query the system can be requested to back-track

and seek other solutions; if the database is unique, for a given set of condition values, it should yield no other requirement. This interactive querying of a proposed regulations database is, in effect, a simulation of the requirements in use. It reveals the behavioural properties of the database and, in this sense, can be regarded as a subjective test of correctness. It confirms that the database behaves as intended and does not yield unexpected results. The testing process provides assurance that the meaning and intent of a set of requirements has been properly conveyed in the expression and organization of the rules.

Other properties of a database can become evident during testing. For example, self-referencing in definitions can be detected, that is, where the value of a rule condition depends upon the outcome of the initial query. This circularity is often difficult to detect in conventional text or even a declarative reading of the database and only becomes apparent in application.

The interactive testing of a database described above is essentially empirical in that any given conclusion depends upon the particular set of data values supplied. Formal tests for the logical properties of a database can be devised which are independent of any one data set and rest upon an analysis of the structure of the database clauses themselves. These tests constitute integrity constraints which the rules and facts, viewed as data structures, must satisfy. The tests exploit the logical mapping possible between the nested list structure of PROLOG clauses and decision tree and information network representations. Thus, in terms of a decision network, one test for completeness is to ensure that in traversing the network there are no rule conditions with only one branching, that is admitting only one data value. Similarly, it is possible to test for uniqueness by determining that all paths through a network are mutually exclusive. Whilst completeness and uniqueness are important properties of a regulations database from an applications point of view there are other logical properties which are equally desirable and for which the database can be tested. For example, it is important for efficiency that there is no redundancy in the database, either in terms of identical rules or in terms of rule conditions whose values are immaterial to the outcome of the rules in which they are embedded.

An important side-effect to this testing of a rule database is the ability to record a trace of a computation, that is, the network of rules and facts traversed in response to some query or test. In the empirical, interactive testing of the database this trace provides a logical proof or explanation of the response to a given query. A trace derived from a formal analysis of the rule and fact clausal structure, on the other hand, reveals the dependencies and hierarchies inherent in the database. These traces of process and structure can be of value when ordering, indexing and cross-referencing regulation requirements in conventional text form.

3.0 Logic programs and expert systems

The interactive testing of a regulations database, in which the system solicits data from the user and displays traces of the computation, has all the characteristics of an expert system. Many of the advantages claimed for expert systems, the explicit representation of knowledge, the consultative dialogue and the explanation of reasoning, are in reality advantages of the rule-based languages in which they are implemented. Logic programming provides an ideal tool for implementing such systems as it inherently separates the representation of knowledge, using facts and rules, from its processing, using logical inference. In turn, building regulations, codes and standards are ideal candidates for expert system as the knowledge they contain is explicit and well documented.

The specification and testing of regulatory material in a logic programming environment would yield ready-made knowledge-bases for use in expert systems. This is seen as a major advantage of the methods advocated here. Using currently available expert systems technology, applications are feasible which would provide end-users with a more convenient and rapid access to regulation requirements than is possible with conventional documentation. More ambitious applications can be envisaged. A regulations expert system interfaced with a suitable modelling system would enable the automatic checking of design proposals for compliance with requirements. Such applications would probably require meta-level knowledge for reasoning with regulations rules and facts rather than simply in them.

4.0 Summary

The paper has described the use of interactive logic programs as a development environment for the formulation and evaluation of building regulation proposals. Logic programming provides a number of important advantages for this work. Regulations information encoded in clausal form in a logic program can be regarded as a concise specification of a proposed requirement. The program can be run, interactively, in any number of test contexts and its behaviour observed. In this way it is possible to determine, empirically, that the proposed regulation will yield, in practice, the result intended by its author. In addition, the inherent structure of logic programs allows the development of formal, computer-based, proofs of the properties of regulation specifications. The databases developed in this way can be used directly as knowledge-bases in expert systems applications.

References

1. I.K. Davies, Get immediate relief with an algorithm, Psychology Today 3,(11), pp.53-69, (1970).

2. B.N. Lewis, I.S. Horabin and C.P. Gane, Flow charts, logical trees and algorithms for rules and regulations, Centre for Administrative Studies Occasional Paper No. 2, (Her Majesty's Stationery Office, London, 1967)

3. B.N. Lewis, Decision logic tables for algorithms and logical trees, Centre for Administrative Studies Occasional Paper No. 12, (Her Majesty's Stationery Office, London, 1970)

4. D.J. Nyman and S.J. Fenves, An organization model for design specifications, J. Struct. Division, American Society of Civil Engineers, 101, pp. 697-716, (1975)

5. J.R. Harris, Logical analysis of Building Code provisions, Proc. 1st NBS/NCSBCS Joint Conference on Research and Innovation in the Building Regulatory Process, (National Bureau of Standards, Washington D.C., 1976), pp. 285-316

6. S.J. Fenves, K. Rankin and H.K. Tejuja, The structure of building specifications, NBS Building Science Series No.9, (National Bureau of Standards, Washington D.C. 1976)

7. S.J. Fenves and R.N. Wright, The representation and use of design specifications, NBS Technical Note 940, (National Bureau of Standards, Washington D.C., 1977)

8. F.I. Stahl, R.N. Wright, S.J. Fenves and J.R. Harris, Expressing standards for computer-aided building design, Journal of Computer-Aided Design 15, (6), pp. 329-334, (1983)

9. J.S. Gero et al., AMUBC system final report, Computer Applications Research Unit, Dept. of Architectural Science, University of Sydney, (1984)

10. K.L. Clark and F.G. McCabe, micro-PROLOG: Programming in logic, (Prentice/Hall International, 1984)