
Chapter 9

An overview of knowledge engineering and its relevance to CAAD

John S. Gero

9.1 Introduction

Computer-aided architectural design (CAAD) has come to mean a number of often disparate activities. These can be placed into one of two categories: using the computer as a drafting and, to a lesser extent, modelling system; and using it as a design medium. The distinction between the two categories is often blurred.

Using the computer as a drafting and modelling tool relies on computing notions concerned with representing objects and structures numerically and with ideas of computer programs as procedural algorithms. Similar notions underly the use of computers as a design medium. We shall return to these later. Clearly, all computer programs contain knowledge, whether methodological knowledge about processes or knowledge about structural relationships in models or databases. However, this knowledge is so intertwined with the procedural representation within the program that it can no longer be seen or found.

Architecture is concerned with much more than numerical descriptions of buildings. It is concerned with concepts, ideas, judgement and experience. All these appear to be outside the realm of traditional computing. Yet architects discoursing use models of buildings largely unrelated to either numerical descriptions or procedural representations. They make use of knowledge - about objects, events and processes - and make nonprocedural (declarative) statements that can only be described symbolically. The limits of traditional computing are the limits of traditional computer-aided design systems, namely, that it is unable directly to represent and manipulate declarative, non-algorithmic, knowledge or to perform symbolic reasoning.

Developments in artificial intelligence have opened up ways of increasing the applicability of computers by acquiring and representing knowledge in computable forms. These approaches supplement rather than supplant existing uses of computers. They begin to allow the explicit representations of human knowledge. The remainder of this chapter provides a brief introduction to this field and describes, through applications, its relevance to computer-aided architectural design.

9.2 Knowledge engineering

Knowledge engineering is a subfield of artificial intelligence. It is concerned with the acquisition, representation and manipulation of human

knowledge in symbolic form. Human knowledge here is thought of as being reasoning (rather than the simple ability to acquire facts as we might find in an encyclopedia). Just as the Industrial Revolution can be considered to have automated mechanical power, and the computer revolution to have automated calculation, so knowledge engineering automates reasoning.

Feigenbaum (1977) defines the activity of knowledge engineering as:

The knowledge engineer practices the art of bringing the principles and tools of artificial intelligence research to bear on difficult application problems requiring experts' knowledge for their solution. The technical issues of acquiring this knowledge, representing it, and using it appropriately to construct and explain lines of reasoning are important in the design of knowledge-based systems.

The fundamental structure used to represent reasoning and, hence, knowledge is *symbolic inference*. Inference is based on well-established logical principles and has been extended to operate on symbols. The primary advantage of inferencing is that it does not require an *a priori* mathematical theory. It can be used to manipulate concepts. Barr and Feigenbaum (1981), writing about the applicability of knowledge engineering in conceptual areas, state:

Since there are no mathematical cores to structure the calculational use of the computer, such areas will inevitably be served by symbolic models and symbolic inference techniques.

Thus, knowledge engineering is primarily concerned with non-algorithmic approaches to computer manipulation. It is convenient to distinguish between *facts* and *knowledge*. Facts are assertions or statements about an object, event or process and which are accepted as being true. Knowledge is the relationship between facts such that when the knowledge is executed in a particular fact domain a new fact is inferred.

9.3 Knowledge representation

There are a variety of possible ways of encoding knowledge. These include (Winston, 1984):

- (1) State-space representations;
- (2) Logic-based representations;
- (3) Procedural representations;
- (4) Semantic nets;
- (5) Production systems; and
- (6) Frame systems.

Although many of these approaches may be mapped on to each other, the one we shall concern ourselves with here is the one which allows us to directly encode knowledge as inference - namely, logic-based representations.

Logic-based representations make use of a special subset of logic called *predicate logic* or *predicate calculus* (Nilsson, 1980) and within that subset

another subset concerned with first-order predicate calculus restricted to Horn clauses (Kowalski, 1979).

Predicate calculus allows us to write single inferences dependent on many conditions of the form:

A is true
if B is true
and C is true
or D is true

('true' does not mean truth in the veracity sense; rather in terms of the relationship of A to B, C and D).

We could infer that A is true from the above inference statement if B and C or D are true - we call such an inference statement *knowledge* to distinguish it from a *fact*, which is the outcome of the inference process.

We could write down some architectural knowledge as:

X is a kitchen
if X contains a sink
and X contains a stove

Thus we would be in the position to 'know' whether any X was a kitchen based on the truth of the two statements following the 'if'.

More formally, such inference statements are called *predicates*. The statement before the *if* is called the head of the predicate whilst the statements following the *if* are called the body of the predicate. Thus, a predicate with both a head and a body is knowledge whilst a predicate with a head only is a fact by our earlier definition.

It is convenient at this stage to re-examine the idea of databases. Databases in this context are used to store representations of designs and their performance. For example, we store the results of a stress analysis in a database possibly to be used by a graphics program. Similarly, we can store the facts we have inferred or directly placed into the system in a facts base and the knowledge in a knowledge base. Such bases are different to databases in a number of important aspects.

Facts and knowledge bases can be added to and subtracted from without affecting the integrity of the remainder of the base. The knowledge base is the computer program and we can have inconsistent facts and other programs for the knowledge to work with. Knowledge can become facts after suitable inferencing.

We can place facts explicitly into the system as a predicate without a body, such as 'X is a kitchen'. We can place knowledge into the system using predicates with bodies. This knowledge is both symbolic and manipulatable (it can, of course, include normal arithmetic and algebraic operations). By allowing the knowledge to operate on available facts we can get the system to carry out the inferencing process to infer whether the head of a predicate is true. If it is, it becomes an inferred fact and can be stored in the facts base.

Thus, we can store both knowledge explicitly (as inferences) and facts. The designer can evaluate the knowledge for correctness, i.e. does it represent what he or she wants it to represent? More likely, the designer uses the system to infer new facts and uses his or her professional

judgement to evaluate these facts in order to extract the design meaning from them.

9.4 Prolog

Whilst it is possible to write logical inferences in such third generation procedural languages as Fortran, Basic, Pascal or Ada, the effort to do so is very high. What is needed is symbolic programming languages. However, there is one symbolic programming language which directly provides the inference structures - it is called *Prolog* (Clocksin and Mellish, 1981).

The general semantics of Prolog predicates are:

```

predicate-name 1 (arguments):-
predicates-name 2 (arguments),
    .
    .
    .
predicate-name n (arguments); predicate-name j (arguments).
    
```

where : means if
 , means and
 ; means or
 . means no more.

The names of the predicates are decided by the programmer. Arguments which commence with an upper case are variables, whilst those which commence with a lower case are values.

Clocksin and Mellish (1981) describe programming in Prolog as:

(1) Declaring some *facts* about objects and their relationships; (2) Defining *rules* about objects and their relationships; and (3) Asking *questions* about objects and their relationships.

Facts are predicates with only a head (i.e. without a body). For example:

```
colour (layer-6, red).
```

in Prolog can be read as the colour of layer 6 is red. Similarly,

```
completed (redesign, part-number-2)
```

in Prolog can be read as redesign of part number 2 has been completed.

Knowledge is a predicate with both a head and a body. For example, we can write down the knowledge we stated earlier needed to determine whether a particular room is a kitchen in Prolog as:

```

kitchen (X):-
    contains (X, sink),
    contains (X, stove).
    
```

What makes Prolog particularly interesting are its following characteristics:

- (1) It carries out symbolic inferencing;
- (2) It can be treated both as a declarative programming language and as a procedural programming language;
- (3) It is a pattern-matching language;
- (4) It uses backtracking in its execution;
- (5) It exhibits non-determinism in its execution; and (6) It can be treated as a non-monotonic language.

9.5 A simple knowledge-based synthesis system

The synthesis of architectural plans can be directly carried out using a knowledge-based system which is modelled on the notion of a language which might be called a *design language*. Such a design language operates by using rules (syntax or grammar) to transform one design state to another. In general these design states can be considered simply to be patterns. This transformation process is called a *production system* (Post, 1943; Stiny, 1980).

We have already seen that logic is a powerful technique for the representation of a wide variety of knowledge. Logic programming, via Prolog, can directly encompass the idea of production systems. What is needed are three classes in our program (Gero and Coyne, 1984):

- (1) A description of the state of the design (facts base);
- (2) A list of transformation rules (rules base); and
- (3) A controller to drive the transformations through the states.

9.5.1 Facts base

The facts base will commence with an initial state and be modified by the application of rules. This base is the design description and is considered simply to be patterns which have meaning for the viewer of these patterns. In Prolog, these facts can be represented as:

```
fact (pattern A).
fact (pattern B).
.
.
fact (pattern Z).
```

9.5.2 Transformation rules

This is the equivalent of the grammar in language and contains the knowledge of how specific patterns can be transformed into other specific patterns. (In English we often hear them expressed in the following form: 'If lighting level too low then replace window by a bigger window'.) In Prolog, these rules can be represented as:

```
rule (1, pattern A --- > pattern AA)
rule (2, pattern B --- > pattern BB)
```

$rule(n, pattern\ ii \dashrightarrow pattern\ jjj).$

The symbol " \dashrightarrow " is a user-defined symbol indicating the idea of transformation.

9.5.3 Controller or interpreter

The control is separated from the transformation knowledge and is in the form of an inference which encodes the instructions for manipulating the facts base. It takes the form of deciding which rules applies; the existence of the appropriate pattern in the design state is checked and if it does exist it is removed and replaced by the new pattern specified by the transformation in that rule. In Prolog this control structure can be represented as:

$control(X):- rule(X, A \dashrightarrow B),$
 $retract(A),$
 $assert(B).$

Where X is the rule label, A and B are of the form *fact (pattern j)* and 'retract' is a Prolog predicate meaning remove from facts base and 'assert' is a Prolog predicate meaning insert into facts base.

Example

Figure 9.1 shows eight transformation rules graphically in the domain of building plans. These have been encoded into knowledge-based system

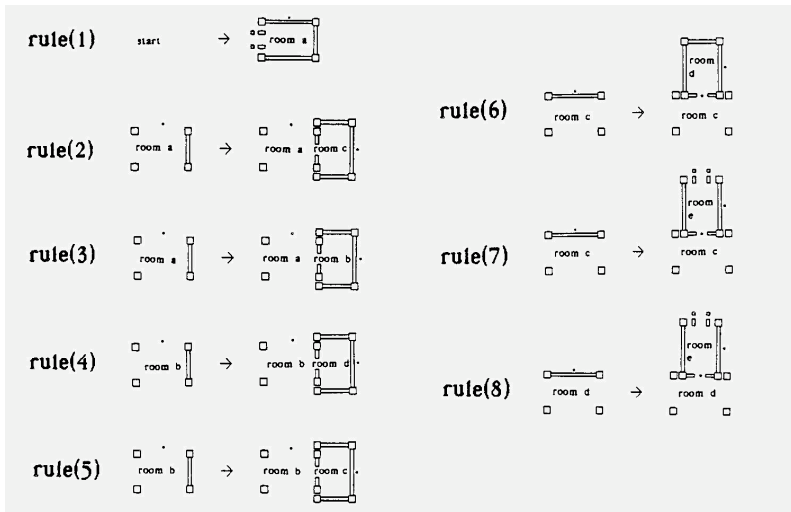


Figure 9.1. The eight transformation rules of a design grammar for synthesizing building plans, shown graphically (Coayne and Gero, 1984)

such as that described above. *Figure 9.2* shows how all the five legal designs are generated by an appropriate application of the rules. It is of interest to note that there is a small finite number of legal designs synthesized. The system is implemented in Prolog on a SUN workstation and all the graphics is direct Prolog output.

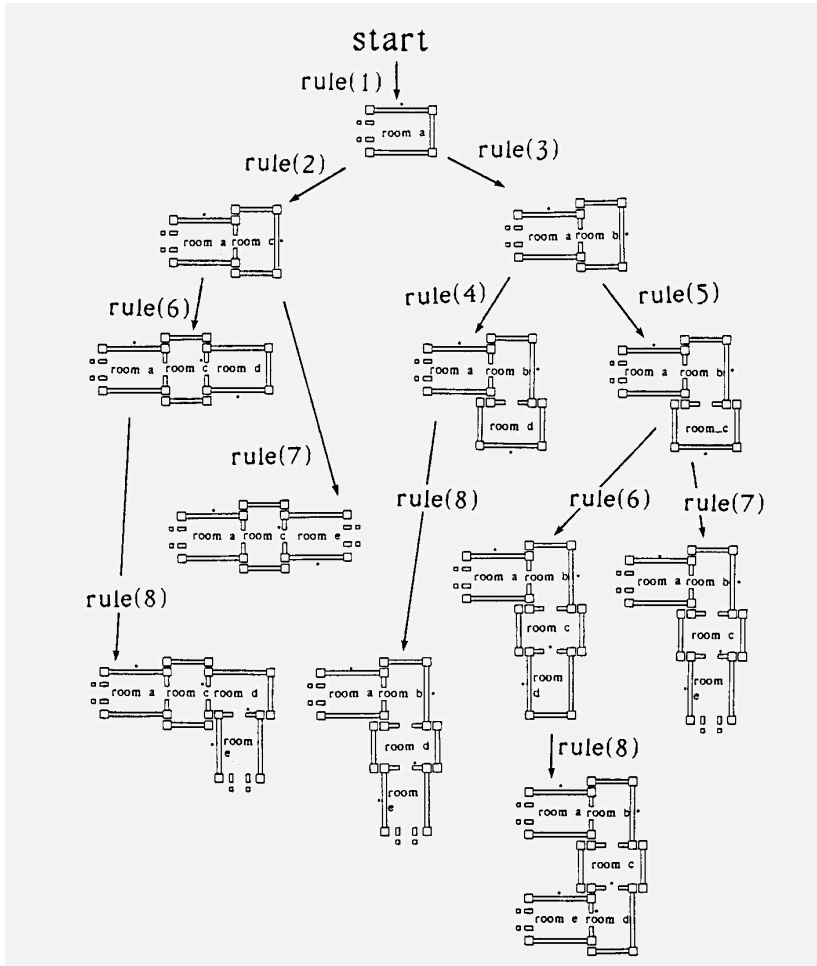


Figure 9.2. The five legal designs synthesized using the rules in Figure 9.1 (Coyne and Gero, 1984)

9.6 Expert systems

Expert systems, which are part of knowledge engineering, have been defined as interactive computer programs which use symbolic inferential

reasoning to deal with problems that are difficult enough to require significant human expertise for their solution. Thus, expert systems aim to capture the ability of rational human experts to ask pertinent questions, to explain why they are asking these questions and to defend their conclusions and recommendations. These characteristics are unrelated to a specific domain of knowledge and apply to all expert systems. All expert systems share a common fundamental structure even if their knowledge encoding mechanisms differ. Each has the following components:

- (1) An inference engine;
- (2) A knowledge base;
- (3) An explanation facility;
- (4) A state description;
- (5) (Possibly) a natural language interface; and
- (6) (Possibly) a knowledge acquisition facility.

The inference engine provides the mechanisms for driving the system. Many shells use a knowledge representation based on production systems or production rules (Buchanan and Shortliffe, 1985). We have already seen that Prolog is a good vehicle for manipulating production systems. Clearly, more complex controllers are needed to allow both data-driven (or forward-chaining) and goal-driven (or backward-chaining) enquiries to be processed. A number of expert-system shells oriented towards computer-aided design have been written in Prolog (Gero, 1985) for rule-based knowledge systems.

Expert system shells (i.e. expert systems without any knowledge in them) can be constructed, such as the BUILD shell (Rosenman, 1985), and then used in a variety of ways, depending on the knowledge base to which they are applied. Rosenman and Gero (1985) have described using the BUILD shell with a knowledge base of the Australian building regulations. A listing of a typical dialogue with the system is reproduced below. Bold after a question mark user input; italics, system output; and horizontal bars, a fact which has been inferred by the system. Asterisks indicate goals either being set by the user or achieved by the system.

```

*****
? find 'fire resistance rating required' given building is hotel.
*****

```

```

-----
building classification is class III
-----

```

```

The number of storeys is?
? why.

```

```

building classification is class III and
number of storeys is greater than 3
needed to prove
fire resisting construction is type 1 and
fire stair requirement

```

The number of storeys is?

?6

fire resisting construction is type 1

fire stair requirement is true

structural member defined?

? how.

The part is?

options for values are:

beam or column or slab

? column.

structural member defined is true

fire resistance rating required is 3 hours

Such systems have the capability of justifying how they arrived at their conclusions. For example:

? explain proof.

*fire resistance rating required is 3 hours
proved true by rule No. r6 because
fire resisting construction is type 1 and
structural member defined is true*

*structural member defined
proved rule by rule No.r3 because
part is column*

*fire resisting construction is type 1 and
fire stair requirement
proved true by rule No. r4 because
building classification is class III and
number of storeys is greater than 3 as
number of storeys is 6*

*building classification is class III
proved true by rule No.r1 because
building is hotel*

BUILD also has the capacity to explain why it did not reach a certain conclusion.

```
*****
? explain why-not 'fire resistance rating required' is none.
*****
```

```
fire resisting construction is type 5
   needed to prove
fire resistance rating required is none
```

```
building classification is class 1
   needed to prove
fire resisting construction is type 5
```

```
building is house
   needed to prove
building classification is class 1
```

So we can see that since the building was a hotel and not a house how the chain of reasoning follows.

Expert systems can be used to generate designs using such a shell as BUILD. In this case the knowledge base consists of knowledge which allows for the continual refinement of prototypical designs. Peter Hutchinson in the Computer Applications Research Unit is constructing a knowledge base which designs retaining walls. *Figure 9.3* shows some of the final output.

Combined synthesis-evaluation systems can be generated with an appropriate knowledge base and the BUILD expert system shell since the shell has both forward- and backward-chaining capabilities. Rivka Oxman in the Computer Applications Research Unit is constructing a system which is the beginning of a designer's apprentice and its example domain is the design of kitchens. It aims to allow the user to construct a design graphically and have the expert system check the design as it proceeds. Alternatively, the user can directly conduct a dialogue with the expert system which then constructs and draws an appropriate design. *Figure 9.4* shows a typical design session.

9.7 Knowledge-based computer-aided architectural design

The encoding of knowledge in a manipulative form is beginning to allow us to address some of the central themes of computer-aided design: How can design synthesis be modelled? What can be automated? What is the knowledge in design processes? The simple knowledge-based design synthesis system of Section 9.5 is inadequate in a variety of ways and more design process control knowledge is needed. This brings us to *planning* systems. Systems which examine the structure of the formulation and generate plans of design processes which when executed produce designs (Gero and Coyne, 1985; Coyne and Gero, 1984; Coyne, 1985).

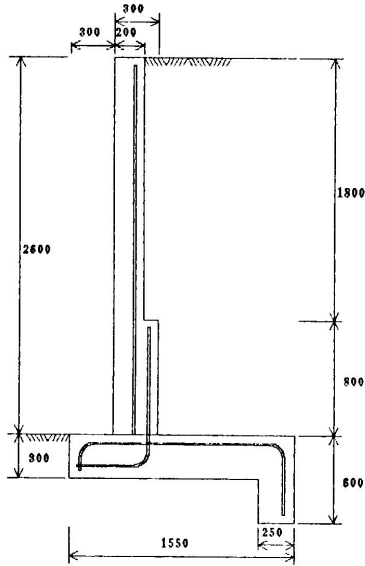


Figure 9.3. The resultant graphical output from the BUILD expert system with a knowledge base for designing retaining walls

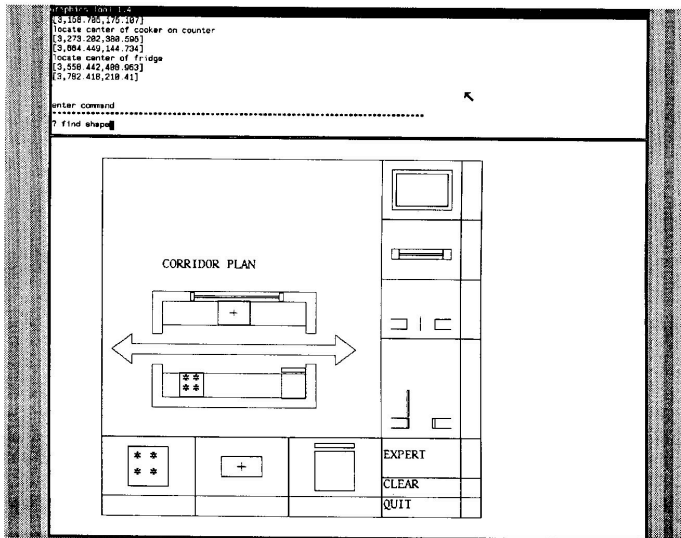


Figure 9.4. A typical design session of a system where the same expert system is used both to check and generate designs graphically

Richard Coyne in the Computer Applications Research Unit has been developing a variety of knowledge-based planning systems. Figure 9.5 shows the results of one such system which utilizes an abstraction of design processes which allow them to be modelled as semantic-syntax maps. Higher-level semantic goals are expanded into lower-level syntactical plans using knowledge. These plans are then criticized by a separate class of knowledge for consistency before that syntactical plan is treated as a sequence of semantic goals and the process repeated until no further expansion is possible. This is the resulting plan of the design process which can be executed to produce the design itself.

9.8 Conclusions

Knowledge engineering provides novel tools in the domain of computer-aided architectural design which supplement rather than supplant our

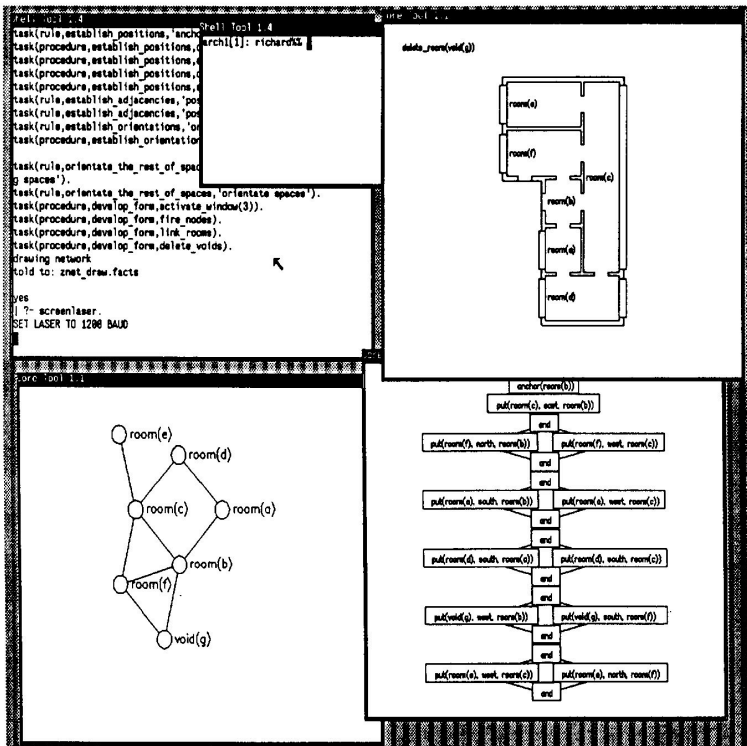


Figure 9.5. The screen dump of a design process planner. The top-left window lists the task the system is executing during the process of constructing a plan. The bottom-left window shows the structure of the design formulation. The bottom-right window shows the completed plan of action which, when executed, produces the design shown in the top-right window

existing tools. However, these tools are opening up avenues and approaches to computer-aided design we were previously unaware of or unable to tackle. Issues central to automated and semi-automated synthesis can now be addressed and computationally tractable systems produced. Whilst knowledge-based systems are certainly not a universal panacea they provide new directions for computer-aided architectural design which expands the role of the computer in design.

Acknowledgements

The knowledge-based systems group in the Computer Applications Research Unit is currently composed of John Gero, Mike Rosenman, Richard Coyne, Bala Balachandran, Conrad Mackenzie, Rivka Oxman, Catherine Manago, Peter Hutchinson, Doug Coates, Richard Leavers and Tony Radford. Research support is from the Australian Research Grants Scheme, the Australian Computer Research Board and the National Energy Demonstration and Development Program.

References

- BARR, A. and FEIGENBAUM, E. (1981). *Handbook of Artificial Intelligence, Vol. 1*, Los Altos: William Kaufmann
- BUCHANAN, B. and SHORTLIFFEE, E. (1984). *Rule-Based Expert Systems*, Reading, Mass.: Addison-Wesley
- CLOCKSIN, w. and MELLISH, c. (1981). *Programming in Prolog*, Berlin: Springer-Verlag
- COYNE, R. D. (1985). A Review of Expert Planning Systems for Computer-aided Design. *Architectural Science Review* 28, No. 4, 95-103
- COYNE, R. and GERO, J. (1984). Design Knowledge and Sequential Plans, *Working Paper*, Computer Applications Research Unit, Department of Architectural Science, University of Sydney.
- FEIGENBAUM, E. (1977). *The Art of Artificial Intelligence: Themes and Case Studies in Knowledge Engineering*. IJCAI-77, Los Altos: William Kaufmann, 1014-1029
- GERO, J. S. (1985). Expert Systems in Design and Analysis. *Nat'l Eng. Conf.*, IEAust, pp. 211-217
- GERO, J., and COYNE, R.D. (1984). Logic Programming as a Means of Representing Semantics in Design Languages. *Working Paper*, Computer Applications Research Unit, Department of Architectural Science, University of Sydney
- GERO, J., and COYNE, R.D. (1985). Knowledge-based Planning as a Design Paradigm. *Working Paper*, Computer Applications Research Unit, Department of Architectural Science, University of Sydney
- KOWALSKI, R. (1979). *Logic for Problem Solving*, Amsterdam: North-Holland
- NILSSON, N. (1980). *Principles of Artificial Intelligence*, Palo Alto: Tioga Publishing
- POST, E. (1943). Formal Reductions of the General Combinatorial Decision Problem. *American J. Maths* 65, 197-268
- ROSENMAN, M. A. (1985). BUILD expert system shell. *User Manual*, Computer Applications Research Unit, Department of Architectural Science, University of Sydney
- ROSENMAN, M. A. and GERO, J. S. (1985). Design Codes as Expert Systems. *Computer-Aided Design* 17, No. 9
- STINY, G. (1980). Introduction to Shape and Shape Grammars. *Environment and Planning B* 7, 343-351
- WINSTON, P. (1984). *Artificial Intelligence*, Reading, Mass.: Addison-Wesley