Argument-based explanation of logic programs

T J M Bench-Capon, D Lowes and A M McEnery

The paper argues that a satisfactory explanation of a logic program must take the form of an argument, rather than a proof. This can only be done on the basis of information regarding the role of the various literals in the bodies of the clauses, which is normally not captured by such programs. A schema for arguments, derived from Toulmin, is presented, and the components of this schema are related to the roles of literals in the bodies of clauses. A metainterpreter is described that uses annotations of body literals to build up an argument structure according to this schema. This structure can then be used to present the argument in a variety of ways; this is illustrated by a discussion of how the argument structure can be used as the basis of a presentation as a paragraph of text. A simple example from a quasilegal domain is presented.

Keywords: explanation, arguments, rule-based programs, logic programming, knowledge-based systems

One of the major attractions of logic programs and expert systems based on the predicate calculus paradigm is their potential for justifying and explaining the conclusions they reach. The standard form of explanation offered is produced by an ascent or descent of the goal tree. A conclusion drawn by the system is based on some clause that licenses that conclusion, and a demonstration of the truth of the literals in the body of the clause. Thus, given a clause of the form

C1: S if P & Q & R

an explanation of S would be expected to be something like

By C1 I can show S if I can show P and Q and R
I can show P
I can show Q
I can show R

If desired P, Q and R can be explained in a similar fashion, until things are reached that are true by virtue of being facts in the knowledge base or input from the user. Such explanations have the attraction that they are a faithful reflection of the reasoning of the system, and that they are simple to implement by a keeping of a record of the goal tree. They are particularly helpful when the knowledge base is built as an aid to the detection of incorrect clauses, but they are not so helpful to the end user, who may well not conceptualize the knowledge base into a chain of clauses, and who, in any event, has no need for this level of detail. Such explanations have, therefore, been castigated as unsatisfactory, as not resembling the kind of explanation that is appropriate for a user of such a system, who wants something much more like the kind of explanation that he/she would receive from an expert, and not a simple recapitulation of the reasoning process. Consider the following PROLOG clause:

```
C2 old(X) :- man(X),
             age(X,A),
             A > 70,
             not tibetan(X).

% all men over 70 are old, except tibetans
% who are exceptionally long-lived
```

Here, the explanation that the system can show that John is old because it can show that John is a man, John has an age of 80, and 80 is greater than 70, and because it is unable to show that John is Tibetan, is clearly not what the user wanted; that John is aged 80 is enough of an explanation for most people, and that any man over 80 is old would be enough of a supplementary explanation to satisfy most of the others. The fact that John is not a Tibetan is unlikely to be mentioned, unless there is some reason to suppose that he is, or unless the person seeking the explanation raises the issue. This kind of explanation is unsatisfactory, because it makes no distinction between a *proof*, which cites a set of premises of which the conclusion is a logical consequence, and an *argument*, which is a presentation of the reasoning that is sufficient to convince the recipient of the argument. When explanations of reasoning are sought outside the formal contexts of mathematics and formal logic, what is looked for is an argument, and not a proof.

The need to present the reasoning in an effective

Department of Computer Science, University of Liverpool, PO Box 147, Liverpool, UK
Paper received 13 July 1990. Revised paper received 16 May 1991

 0950-7051/91/030177-07

manner shows itself in a variety of ways. The argument tends to omit some of the premises that it is supposed are already accepted by the audience; in the case above, it is usually accepted by both parties that John is a man and is not a Tibetan, and so these premises need not appear in the argument. Also, there is some combination of premises: in the example, age(john,80) and 80 > 70 need to be taken together to give something like the age of John is greater than 70. Thus an argument, and, hence, a satisfactory explanation, needs an intelligent consideration of the premises and their contributions to the argument, so that this kind of selection can be made. This contrasts with the proof, which treats all the premises as being of equal status.

If C2 is returned to, and the literals in the body of the clause are examined, it can be seen that they are, in fact, there for different reasons. The first literal, man(X), provides information as to the sortal concept under which the individual must fall if the rest of the clause is to be applicable; if dogs were being considered, the age limit for being old would be very much lower. The second literal, age(X,*A*), is required to retrieve the age of the individual from the database. Note that it is not expected to fail, so that its truth is not in question, but it is needed to instantiate *A*. The third literal, $A > 70$, provides a test of the value of *A*, which is the real truth condition; if the domain of consideration is limited to men, it is the success or failure of this literal that typically determines the value of old(X). The final literal is there to cover rare or odd cases that represent an exception to the general rule.

Considered simply as part of a logic program, these differing motivations for the literals in the body can be ignored, as one is only interested in the relationships between the truth values of the various terms, and the interpretation suggested by the various predicate names is not strictly material. However, the interpretation returns to the fore when the clause is used as part of a system that explains its reasoning, and the different motivations critically affect what is a sensible explanation with regard to the interpretation. The upshot is this: by concentration on the logic, a program is produced that is adequate for the correct conclusions to be drawn, but some information is lost, i.e. the information pertaining to the role of the various literals, and it is this extralogical information that is required for an argument or explanation, as opposed to a proof, to be produced. It is, therefore, necessary to include this information in some form if a satisfactory explanation is to be given.

One very simple approach, previously adopted by Bench-Capon and McEnery[1], is to include this information as a template that represents the desired form of presentation, which is held as an additional argument in the head. Thus, C2 would become

```
C2b old(X,[X,is,old,because,all,men,
        aged,over,70,are old,unless,
        they,are,tibetan,and,X,is,A,years,old]) :-
            man(X),
            age(X,A),
            A > 70,
            not tibetan(X).
```

When a body literal is itself deduced, it too is supplied with an explanation template that is located appropriately in the explanation template of the head. Thus, if man(X) were defined by some other clause in the database, C2b would become

```
C2c old(X,[X,is,old,because,all,men,aged,over,70,
        are old,unless,they,are,tibetan,E,and,
        X,is,A,years.old]) :-
        man(X,E),
        age(X,A),
        A > 70,
        not tibetan(X).
```

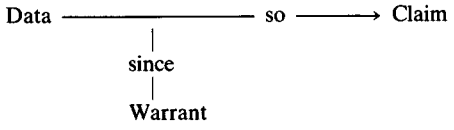
where E would become instantiated to a suitable explanation of the truth of man(X), enabling this to be included in the explanation of old(X). This method can produce effective explanations, particularly if they are postprocessed to break up what can, in lengthy inference chains, become unreadably long sentences into more presentable prose. This approach allows the explanation for every clause to be customized, so giving a very precise account of the role of the various body literals, but it has the obvious drawback that it is necessary to write the appropriate template for every clause. What would be preferable would be to have a generally applicable formalism for an argument with a link between this formalism and the extralogical role of the body literals. This would enable the different motivations for the inclusion of these literals to be recorded in the form of simple annotations, a proof to be constructed without reference to these annotations, and the annotations to then be exploited, via the argument formalism, in the presentation of the reasoning. This approach ensures that the explanation has the information that it needs, while obviating the need to provide specific explanation templates for every clause. The implementation of this approach is the subject of this paper.

Critical to the approach is the adoption of a schema for representing arguments. One popular schema is that developed by Stephen Toulmin[2] and adopted by a number of researchers, particularly in the field of artificial intelligence (AI) and the law. The Alvey and UK Department of Health and Social Security Demonstrator project[3], and the Dick[4], Marshall[5] and Lutomski[6] papers may be cited as examples of systems for which the schema has proved helpful and effective for representing arguments. The next section, therefore, introduces Toulmin's formalism, presents some extensions to it, and shows how annotations indicating the role of literals in the bodies of clauses relate to that schema.

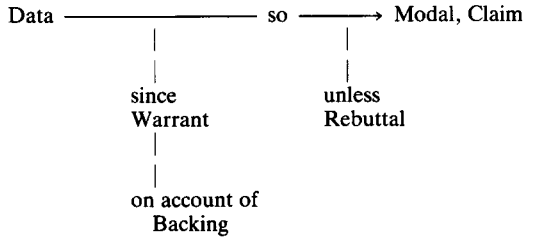
TOULMIN'S ARGUMENT SCHEMA

Toulmin's schema for argument representation is described in his book *The Uses of Argument*[2]. Toulmin's representation is intended to provide a framework for the analysis of jurisprudential and other types of argument, while maintaining the procedural and declarative aspects of such arguments.

The argument schema is developed from an initial assertion (or claim) that must then be justified by facts (or data) being called upon. The step from data to claim must also be justified by the rules (or warrants) that have been used. The representation can be shown diagrammatically.



Toulmin continues the development of the representation by including the notions of backing, to provide further justification for the warrant, rebuttals, to allow for exceptional conditions (conditions under which a *prima facie* argument may be rebutted), and modal qualifiers, to allow for variations in the degree of force of the argument. The resulting schema is as follows:



Schemata such as Toulmin's have proved to be popular for giving a well structured and clear form for explanation. The authors' aim was to develop such a schema to provide a model for explanation in logic programs that could then provide a basis for presentation of the explanation in a variety of ways, either in the form of diagrams, as above, or, as discussed below, as paragraphs of text.

The method used was that of identifying the extra-logical role played by each part of the clause, and how the various roles related to Toulmin's argument components. Toulmin's schema already provides something of what is required. If a clause is taken as expressing a simple argument for the truth of the head of the clause, Toulmin's schema can be used to distinguish between three kinds of premise: the basic conditions that form the data, the operative clause, which forms the warrant, and the exceptional conditions that form the rebuttal. This would, however, create a restriction to a single argument/clause, whereas the typical explanation of a logic program involves a chain of such small arguments. The Toulmin schema was therefore extended to allow for the justification of data by the creation of further arguments, so that a chain of arguments could be formed where the claim of one argument could be a datum for the next, reflecting the chaining of clauses in a logic program. Second, it must be noted that Toulmin, who is concerned with arguments expressed in a natural language, typically includes warrants of the form 'all men over 70 are old'. As an untyped logic is used to generate the authors' arguments, an unmodified use of Toulmin's method would require that data such as 'John is a man' be had. However, this moves this part of the argument from its correct location, as it justifies the warrant, and not the claim. Two components were therefore added to the Toulmin schema: 'context' and 'basis'. These allowed the sortal information to be located appropriately, by justifying the applicability of warrants by appealing to the sort of the individual under consideration. The sortal information was termed a 'context', because such information is normally taken as understood when the singular term is used in the query; it is arguable, indeed, that the use of a proper name is illegitimate if

Table 1. Roles identified with elements in argument schema

Role in clause	Literal	Role in Argument
Claim	old(X)	The head of the clause corresponds to the claim
Class	man(X)	Defines the things for which the warrant is applicable; it corresponds to the context
Data	age(X, *A*)	Data used to justify the claim
Cond	greater_than(*A*, 70)	Condition on a datum, and part of the warrant
Qual	not(tibetan(X))	Defines an exceptional condition when the clause is not applicable; it corresponds to a rebuttal

the user does not know under which sortal the thing named falls. The basis is the warrant extended to include the contextual information.

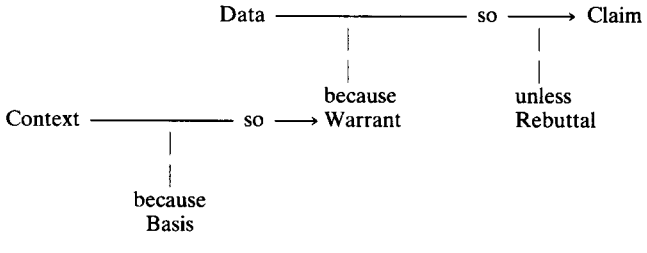
In the authors' example of the clause for old, the specific age condition may well depend on the type of individual:

a man is old if aged over 70,
a dog is old if aged over 10 etc.

In this case, the sortal (man or dog) is called part of the 'context', and the warrant becomes 'old if aged over 70' in the case of a man, and 'old if aged over 10' in the case of a dog. The 'basis' is 'old if man and aged over 70', which is an expanded version of the warrant, showing the effect of the sortal information.

Omitted entirely as being inapplicable to standard logic programs were Toulmin's modal qualifiers (which have no counterparts in 1st-order predicate calculus), and backing, which, although important for Toulmin's schema, has no obvious counterpart in a logic program. These omissions mean that the full richness of Toulmin's schema cannot be captured, but they do not vitiate the authors' more limited aims, which are to exploit some of his ideas to help organize their explanations of logic programs.

The authors' modified schema, intended to be a general form of argument represented by the successful application of a clause in a logic program, is thus



Of course, some of these elements may not be present in a given case.

It is now necessary to identify the role played by each part of a clause in the explanation, and the relationship of the role to the Toulmin schema. Consider the example of C2 used above, and recall the four different roles played by the body literals.

Those roles were identified with elements in the authors' argument schema giving rise, in the example in this paper, to the information in Table 1.

The following warrants were identified for the authors' schema:

Warrant old(X) if age(X,*A*)and greater_than(*A*,70)

Note that this is the clause without class and qualification. This corresponds to the authors' observation that rules are typically expressed in English in terms of a sortal concept, e.g. 'all men are old if they are over 70', and not 'all things are old if they are men and over 70'.

Basis old(X)if man(X),age(X,*A*)and greater_than(*A*,70)

Note that this is the clause without qualification. Rules are normally expressed in a general but defeasible way; the defeasibility is only considered when an exceptional case is suspected.

ARGUMENT PROGRAM

The different extralogical roles in the logic programs and the relationship of each role to the modification of Toulmin's schema having been identified, the next step was to design an annotated form for the logic program, and a metainterpreter to transform proofs into the argument schema.

The clauses in the program are annotated by the addition of an extra argument to each literal in the body of each clause. This argument is an atom, and must be one of {data,cond,qual,class}; it designates the role that each literal plays. The annotated version of C2 appears as

```
C2a   old(X):-man(X,class)
        age(X, A, data),
        greater_than( A, 70, cond),
        not( tibetan( X ), qual).
```

Note that the head goal in a clause can play different roles in different arguments (e.g. old(X) may be any of data, cond, class or qual). However, in relation to the body of the clause, the head always plays the role of claim, and so does not need to be annotated.

The interpreter is a backward-chaining prover in which the 'call' predicate has been specialized to handle the annotated subgoals. The interpreter is given a claim (or goal), and builds a proof trace in the form of the schema components; the components from this trace are then asserted in relational form with unique argument identifiers to indicate to which argument (or subargument) each component belongs. This relational form can then be manipulated to give the explanation in the desired form. The argument identifiers are generated automatically by the interpreter.

As an example, suppose that C2a is executed, with the additional clause

```
C3a   man( X ):- human(X, class),
                 male( X, data).
```

and the fact

```
F1    age( john, 80).
```

The call to the interpreter is make_argument(old(john)).

The interpreter then displays

```
Proved old( john ) with arguments [a1, a2].
```

The relational forms of the arguments a1 and a2 are as follows:

```
argument( a1 ).
argument( a2 ).

claim( a1, old( john )).
claim( a2, man( john )).

data( a1, age( john,80)).
data( a2, male( john, data)).

rebuttal( a1, not( tibetan( john))).

warrant( a1, [old(john),if,age(john, X), greater_than(X, 70)]).
warrant( a2, [man(john), if, male( john )]).

context( a1, man( john ), a2).
context( a2, human( john )).

basis( a1, [old(X), if, man(X), age(X,Y), greater_than(Y,70)]).
basis( a2, [man(X), if, human(X), male(X)]).
```

In the case of context, there can be a third argument, which indicates that the context of its argument is further defined by the argument a2.

When trying to prove a claim at any stage, the interpreter checks through the existing arguments. If an argument for the claim already exists, then there is no need to reprove it; a cross reference to the existing argument is inserted into the proof trace.

Thus the metainterpreter uses the annotations in the clauses and its proof trace to assert a number of relationships that together constitute a description of the modified Toulmin structure for the chain of argument that confirmed the query. This can then form the basis for the explanation to be given to the user. Thus, given the above example, Figure 1 might be produced.

This figure would be suitable for someone who understood the argument formalism, and was unafraid of the relational expressions. The authors believe that it contrasts favourably with the 'proof' explanation. However, it is believed that a more effective and universally convincing explanation would be produced by the use of the argument formalism to generate an appropriate paragraph of text. This form of presentation is discussed in the next section.

USING A TOULMIN STRUCTURE TO GENERATE A TEXTUAL EXPLANATION

The form of output required to be generated from the Toulmin structures is a passage of English text presenting the argument. The generation of such a paragraph is achieved by three steps. First, a straightforward series of templates is used to generate simple sentences from the predicates. This is little more than syntactic 'sugar', although it is very helpful sugar for the user, but the second, and far more important, step uses the fact that the Toulmin argument structure itself constitutes a plan for output, and can therefore use work on natural-language generation using plans, a common and effective method of producing natural-language output. Finally, the presentation is further refined by the introduction of appropriate pronouns. In this section, each of these steps is discussed, and some example outputs are then presented.

The output from the Toulmin metainterpreter is in the form of the series of relationships describing the argument. The natural-language module uses only a knowledge of these relevant relationships to promote data abstraction. The method of converting the predicates to simple sentences uses the simple, but effective, technique reported by McEnery[7]. The technique

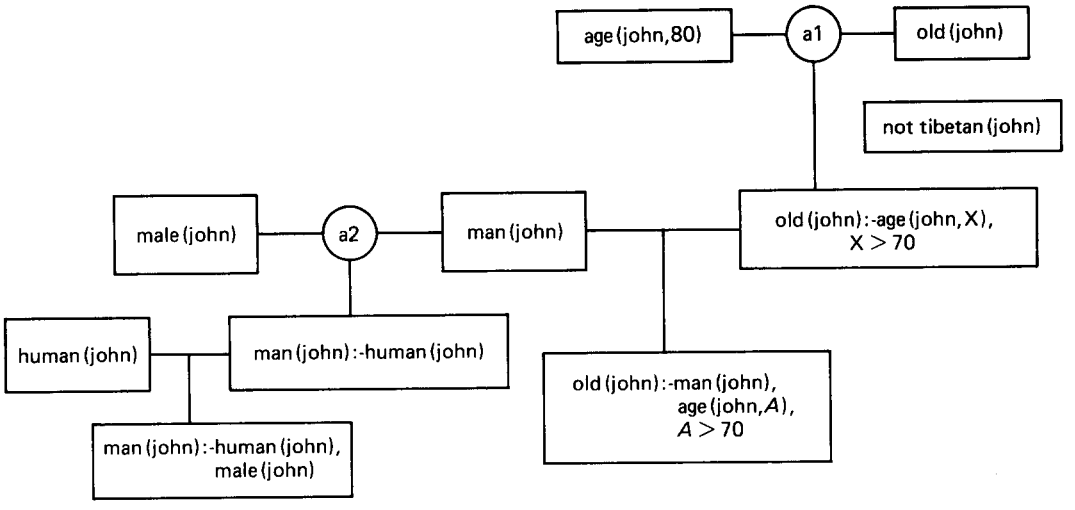


Figure 1. Basis of explanation for user

depends on the introduction of a linguistic typing for each of the elements to be output. In, for example, a PROLOG program, each of the predicates used receives a linguistic typing, and the output level is generally one literal to one sentence. Several literals may be output in one sentence, however, where they are conjoined within the body of one main clause. The linguistic typing of each predicate determines what skeleton sentence type is used to output the literal in question. As an example, one type of predicate is *assertive*. Assertive predicates essentially transpose onto a skeleton sentence in which a subject is related to a complement via the verb 'to be'. Hence, the basic sentential skeleton is of the form 'X is Y'. Thus, the formation of sentences in the current output module leads to output based on a series of skeleton sentences that are selected on the basis of some manually generated typing of a predicate. The advantage of introducing this typing information is that it reduces the need for individual pieces of canned text to be provided for each of the predicates; instead, it uses information that corresponds to part of a speaker's knowledge of how a word is to be used.

This technique could be used on its own to make any explanation more presentable, but this would still not address the need to treat the sentences differently according to their role when it comes to the presentation of the argument. The actual coherence given to the output sentences is afforded by the use of the Toulmin schema, and it is this that is most worthy from the natural-language processing point of view in the current work. Much effort has been expended within computational linguistics, not merely to generate sentences, but also to make those sentences cohere in an interrelated whole that may be called a text. One popular method of achieving this has been the use of plans, as used, for example, by Wilensky[8], and other schema theoretical constructs. The authors' earlier work involving the use of plans to generate natural-language descriptions of logic programs is described in Reference 3. In the CHRISTIE program, after an initial attempt to provide descriptions from templates of canned text, a move was made towards the generation of descriptions from schema theoretical plans. A program was described in schema theoretical terms, and the program examined these stored relationships in explaining its reasoning, much as the current program uses the stored Toulmin relationships. The main problem with this approach was, however, that a lengthy period of analysis was required for each program, as the schema of each program had to be generated manually. The Toulmin argument structure is clearly related to schema theory, but it has some significant advantages. First, the Toulmin structure is composed of a small number of elements, and is well specified. This is in contrast to schema theory in general, within which no common grounds exist to describe what constitutes a schema, or of what formal elements it is composed, as was shown by Cavilcanti[9]. Second, the Toulmin argument structure deals only with a specific subset of schema theory that is relevant to logically ordered conversation. The explanation of logic programs is a single type of activity, and so it should be susceptible to being driven by a single general plan, which the modified Toulmin framework can provide. For this reason, it is over complicating matters to use the whole of whatever may constitute schema theory to describe the output from such programs. Hence, from a natural-language generation point of view, the current work is of some significance, as it not only shows that output at the sentential level can be achieved with a low level of cost, but it also suggests that, in the generation of text from plans, the use of specific models such as the Toulmin argument may be more fruitful in certain application areas than that of other models that, although ostensibly more powerful, are overgeneral in terms of meeting the needs of the application economically.

The final stage, that of pronominalization, uses the techniques described in Reference 10.

The actual form of the output is now examined. The plan is to provide three sentences. The first will contain only the data, and will represent a typically sufficient explanation; the second will contain only the warrant, showing why it is an explanation; the third will contain only the rebuttal, representing the dismissal of exceptional cases. Note that, at this level of explanation, neither the context nor the basis is mentioned, as this

information should be taken as being understood by the user. This gives the following output:

John is old because his age is 80.
John is old if his age is greater than 70.
John is not Tibetan.

If desired, some extra means of interaction could be provided, so that the warrant and rebuttal would be presented only on request, as, as has been noted, the first sentence is often enough of an explanation by itself. Had the contextual assumption been violated, and the user been unaware of the sortal under which John fell, he/she might have been unsatisfied with the second of these sentences. On request, the further explanation, using the context and the basis, can be given:

John is a man.
A man is old if he is over 70.

Where the fact that John is a man can be proven from a further clause, there is an argument to explain the first of these sentences:

John is male
John is a man if he is male

where the context is now only that John is human. This simple example is enough to demonstrate the following points, which result from the ability to assign different roles to the clauses, and, hence, to use them differently when the explanation is presented. First, the contextual sortal information is held back unless it is requested by the user. Second, the exceptional case is separated from the main condition so that attention is drawn to the key factor in the explanation. Third, the two literals using an internal variable have been combined so that the irritating (because obvious) 80 > 70 no longer appears at all. Fourth, there are now two further ways of obtaining further explanation: either one can chain back through the data elements, or one can ask that the context be made explicit, so obtaining a reason why the clause applies. All of these were included in the *desiderata* for a successful argument outlined in the introduction.

While the simple example is helpful in understanding the mechanics of how the process works, it is too simple to exhibit the features to best effect. Therefore, consider the following example pertaining to (fictional) nationality legislation. Suppose that there is an annotated clause

```
citizen(X,eden):-
        male(X,class),
        father(X,Y,data),
        citizen(Y,eden,cond),
        age(X,A,data),
        greater_than(A,16,cond),
        not( murderer(X),qual),
        not( exiled(X),qual).
```

If this is run with the query citizen(abel,eden), the following explanation would be obtained:

Abel is a citizen of Eden because his father is a citizen of Eden and his age is 23.
Abel is a citizen of Eden if his father is a citizen of Eden and his age is greater than 16.
Abel is not a murderer and he has not been exiled.

This seems to the authors to be an excellent explanation. Those in any doubt should construct the conventional proof-based explanation, and make the comparison.

CONCLUSIONS

This paper has outlined a simple technique for augmenting the information incorporated within a logic program so as to produce acceptable explanations of its conclusions. Central are two types of knowledge: first, the role played by the various literals in the body of a clause used to reach a conclusion, and second, a general notion of how to present an argument for a conclusion. The first is represented by annotations to the clauses, and the second by knowledge of a generalized argument schema used by a metainterpreter. The output from the interpreter is entirely independent of any presentation format. This enables the chosen presentation module to use any desired format, such as a diagram or text, but this module requires, of course, additional information governing the diagrammatic or linguistic conventions to be used. The authors believe that the distinction between a proof and an argument is an important one, and that their work demonstrates part of the considerable mileage that can result from taking the distinction seriously.

ACKNOWLEDGEMENTS

Some of the work described above was carried out as part of the UK Department of Health and Social Security Large Demonstrator Project, which was supported by the Alvey Directorate of the UK Department of Trade and Industry and the UK Science and Engineering Research Council. The project collaborators were ICI, Logica, Imperial College, University of London, and the Universities of Lancaster, Liverpool and Surrey (all in the UK). The views expressed in this paper are those of the authors, and they may not necessarily be shared by the other collaborators.

REFERENCES

1 **Bench-Capon, T J M and McEnery, A M** 'CHRISTIE: Towards explaining logic programs' *Alvey Report 89/3* Dep. Computer Science, University of Liverpool, UK (1989)
2 **Toulmin, S** *The Uses of Argument* Cambridge University Press, UK (1958)
3 **Bench-Capon, T J M** (Ed.) *Legal Applications of KBS* Academic Press (1991)
4 **Dick, J P** 'Conceptual retrieval and case law' *Proc. 1st Int. Conf. AI and Law* ACM Press, USA (1987)
5 **Marshall, C C** 'Representing the structure of a legal argument' *Proc. 2nd Int. Conf. AI and Law* ACM Press, USA (1989)
6 **Lutomski, L S** 'The design of an attorney's statistical consultant' *Proc. 2nd Int. Conf. AI and Law* ACM Press, USA (1989)
7 **McEnery, A M** *Computational Linguistics* Sigma Press, USA (1990)

8 **Wilensky, R** 'PAM' *in* **Schank, R C and Riesbeck, C K (Eds.)** *Inside Computer Understanding* Lawrence Erlbaum, USA (1981)
9 **Cavilcanti, M** 'The pragmatics of foreign language reader–text interaction' *PhD Thesis* Dep. Linguistics, Lancaster University, UK (1983)
10 **McEnery, A M and Bench-Capon, T J M** 'The WEB cohesive tie marking system: an expert system to resolve pronoun reference' *Expert Syst. Inf. Manage.* Vol 2 No 3 (1989) pp 157–171