An Application of the Expert System Shell Σkilagi to the Study Benefits Regulation of Bologna University

Giovanni Dallara
Cirfid, Bologna University ,Via Galliera 3/5, 40100, Bologna.
E-mail: g3wbov21@icineca.cineca.it
Carlo Gattei
Cirfid, Bologna University
Giovanni Sartor
Cirfid, Bologna University

ABSTRACT

Cirfid (Centre of Computer Science and Law of Bologna University) has developed a system that applies the study benefits law of Bologna University. The system is able to establish the benefits a student is entitled to on the basis of his family status and university curriculum, and is intended to be used by both university students and public employees charged with applying this law. To achieve this application we used a prototype version of Σkilagi, a logic programming-based expert system shell developed by Marek Sergot and Yannis Cosmadopoulos at Imperial College, London. Some characteristics of Σkilagi, and particularly the possibility of giving conditional answers, are very useful for this application and for legal problems in general.

We have developed an additional facility of Σkilagi, allowing the utilization of questionnaires in order to simplify user-system interaction.

THE EXPERT SYSTEM SHELL ΣKILAGI

Σkilagi (Cosmadopoulos and Sergot 1991) is a logic programming-based expert system shell developed by Marek Sergot and Yannis Cosmadopoulos at Imperial College, London. It has been implemented in Lpa Prolog and, for this application, the version for Macintosh computers was used.

Σkilagi develops the experience of Apes (Sergot 1983; Hammond and Sergot 1984), a shell already used in some legal applications of logic programming (Sergot et al. 1986, and, in Italy, Andretta et al. 1988).

The principal components of Apes are an interpreter, using the standard Prolog proof procedure, and an interactive component, based on the *query the user* model (the user is considered an extension of the knowledge-base, cf. Sergot 1983). When a goal G cannot be solved on the basis of the program, the interactive component asks the user for information: if G is ground, it asks if it is true (Yes/No questions); if G is not ground, it asks for a variable substitution making G true (Wh questions).

Σkilagi keeps many characteristics of Apes, but it utilizes translation, instead of metainterpretation, extends the Prolog model with constructive negation, and develops the *query the user* facility with conditional answers so as to manage hypotheses.

TRANSLATION

Σkilagi is based on the method of translation (cf. Cosmadopoulos and Southwick 1989), instead of metainterpretation.

Metainterpretation has often been used to develop Prolog based shells, providing an inference engine and a representation language. The inference engine operates on a domain program written in the representation language. The distinction between inference engine and domain knowledge provides modifiability and clarity, but at the cost of a loss of efficiency.

The method of translation, based on blending all metalevel functionalities into the object level, can help to solve the efficiency problem. The domain program is translated into a code that can be directly executed by a Prolog interpreter, taking advantage of its optimization techniques. This *blending transformation* happens automatically during the 'compile-time'.

CONSTRUCTIVE NEGATION

In Σkilagi, constructive negation (Chan 1988) has been implemented. Constructive negation is based on the consideration that, in the completion of the program, the following is a theorem

$$Q \equiv A_1 \vee ... \vee A_n$$

where $A_1, ..., A_n$ are the answers to the positive goal Q, epresented as equations. The rule for constructive negation s, in fact, the following:

$$\sim Q \equiv \sim (A_1 \vee ... \vee A_n).$$

If we apply to ~ $(A_1 \vee \ldots \vee A_n)$ the transformation defined by Chan (1988, 119), we obtain a disjunction $NA_1 \vee \ldots \vee NA_m$, where every NA_i is a normal answer to the negative goal ~ A. A normal answer is constituted by a conjunction of equalities and inequalities.

These inequalities qualify Σkilagi's answers. For example, given the program

```
p(a).
p(X) :- q(X).
q(b).
```

when not(p(X)) is asked, Σkilagi gives the answer

yes

with the qualifications

not(X = a) & not (X = b).

Constructive negation allows non ground negative goals to be correctly executed. So the floundering problem can be solved.

EXPLICITLY QUANTIFIED NEGATION

Constructive negation, by treating correctly negative non ground goals as

$\leftarrow$ not p(X),

distinguishes them from explicitly quantified negative goals (cf. Chan 1988, 121) like

$\leftarrow \forall Y$ (not p(Y)).

So, Σkilagi's language allows explicitly quantified negative goals (or subgoals) to be expressed.

THE SYSTEM-USER INTERACTION

In Σkilagi, as in Apes, the ways to elicit information from the user can be specified through metadeclarations. So, writing an application with Σkilagi includes:

- a logical representation of the domain knowledge (the language offered by Σkilagi to this purpose is pure Prolog);
- a definition of the ways of interaction with the user.

For every predicate we can determine how questions asked by the system will be formulated, by using the metapredicate template/3. This predicate has the structure:

template(<predicate>,<list of variables>,<comment>),

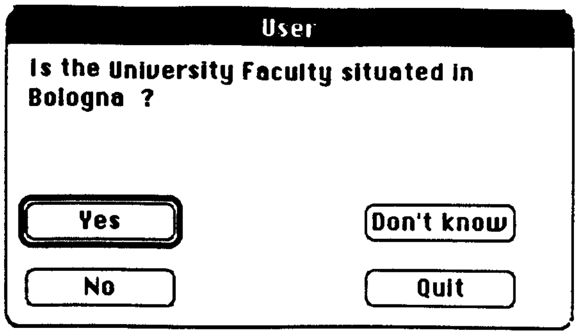
where <predicate> is the name of a predicate of the object program, <list of variables> is the list of the variables of the predicate that have to be instantiated, <comment> is the structure of the question.

In the following example a Yes/No question is defined:

```
template(placed_in(UniversitySchool,City),[],
            [UniversitySchool,' is situated in ',City,'?']).
```

where the empty list indicates that there is no variable to instantiate.

Figure 1. *Yes/No Question*

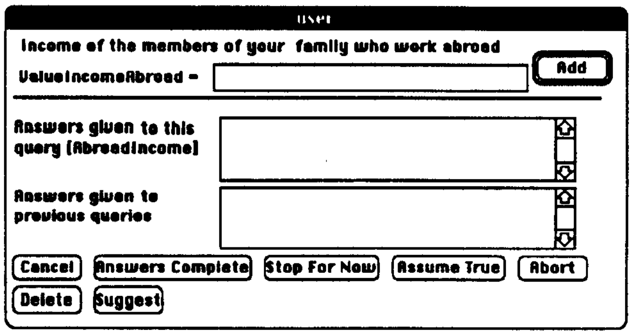


Is the University Faculty situated in Bologna ?

Here a Wh question is defined:

```
template(
    family_income_earned_abroad_of(Name,
                        Surname,ValueIncomeAbroad),
    [ValueIncomeAbroad],
    ['Income of the members of
     the family of ',Name,Surname,'who work abroad']).
```

Figure 2. *Wh Question*



Income of the members of your family who work abroad

ValueIncomeAbroad =

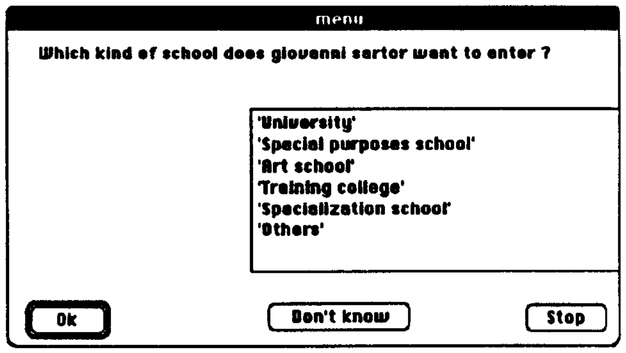
Answers given to this query [AbroadIncome]

Answers given to previous queries

The metapredicate menu_item can be used to confine the user's choices to a predefined set of possible values. For example:

```
menu_item(attends(Name,Surname,UniversitySchool),
      [UniversitySchool],
      ['University','Special purposes school',
             'Art school','Training college',
             'Specialization school','Others'],
      ['Which kind of school does',
                Name,Surname,' want to enter ?']).
```

Figure 3. *Menu question*



Which kind of school does giovanni sartor want to enter ?

'University'
'Special purposes school'
'Art school'
'Training college'
'Specialization school'
'Others'

The user can answer the system questions as follows:

- Yes/No Questions. The user can answer *Yes*, *No*, or *Assume true* if he does not know whether the fact in question is true or false. In this last situation he may get a conditional answer, which is qualified both by the inequalities resulting from constructive negation and by those facts assumed true.

- Wh Questions. If the user knows some true instances of the predicate, he can indicate corresponding values for the predicate variables. Otherwise, he can answer *Don't know*. In which case, he can obtain a conditional answer, qualified by the conditions that the predicate variables must satisfy in order to solve the goal.

THE QUESTIONNAIRES

Developing our application, we realized that, by using the *query the user* facility, the system asked numerous questions, and that correct interpretation of each question presupposed an understanding of its relation with others. So we decided to implement a facility allowing an interaction through questionnaires, each containing a set of related questions. Questionnaires can be defined using the metapredicate menu_form/4, with the following structure:

```
menu_form(<name>,<predicates>,
                    <variables>,<comments>)
```

where <name> is the name of the questionnaire, <predicates> is the list of the predicates of the questionnaire, <variables> is the list of the variables' lists that have to be instantiated through the questionnaire (one list for each predicate), <comments> is the list of the questions included in the questionnaire.
Here is an example of a questionnaire declaration:

```
menu_form('Economic Requirements',
  [special_indemnity(Name,Surname,Indemnity),
   number_of_family_members_of(Name,
                               Surname,NumFam),
   unemployed_children_in_the_family_of(Name,
                               Surname,SonsDaughters),
   already_registered_in_assistance_file(Name,Surname),
   has_family_residence(Name,Surname,'Italy'),
   has_family_member_working_abroad(Name,Surname)],
  [[Indemnity],[NumFam],[SonsDaughters],[],[],[]],
  [['Special family indemnity of ',Name,Surname],
   ['Number of family members of ',Name,Surname,' ?'],
   ['Unemployed children in the family of ',
                               Name,Surname,' ?'],
   ['Is ',Name,Surname,' already registered in the
                     University Assistance Files ?'],
   ['Is the family residence of ',Name,Surname,'in Italy ?'],
   ['Are there members of the family of ',Name,Surname,
                               'working abroad ?']]).
```

The questionnaires are activated with the predefined predicate

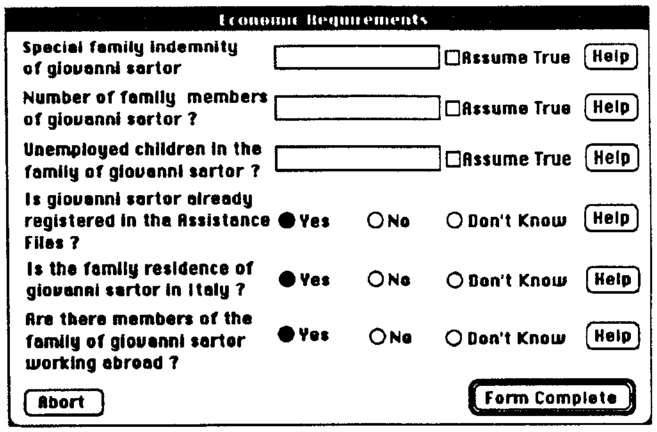
```
form(<name>,<input>)
```

where <name> is the name of the questionnaire and <input> is a list of input values for the predicates in the questionnaire. The questionnaire will not include questions concerning variables for which input values have been passed. Those values may be used inside comments. For example, if the questionnaire defined above is activated with the call

```
form('Economic Requirements',[Name,Surname]),
```

the variables Name and Surname being instantiated to the values 'giovanni' and 'sartor', the following is displayed.

Figure 4. *Questionnaire 'Economic Requirements'*



Economic Requirements

Special family indemnity of giovanni sartor		☐ Assume True		Help
Number of family members of giovanni sartor ?		☐ Assume True		Help
Unemployed children in the family of giovanni sartor ?		☐ Assume True		Help
Is giovanni sartor already registered in the Assistance Files ?	[✓ Yes]	No	Don't Know	Help
Is the family residence of giovanni sartor in Italy ?	[✓ Yes]	No	Don't Know	Help
Are there members of the family of giovanni sartor working abroad ?	[✓ Yes]	No	Don't Know	Help

A 'Help button' is associated with every question on the questionnaire. By pushing the button, the user can obtain more detailed information about the question.

THE APPLICATION TO THE STUDY BENEFITS LAW

The study benefits law applied by Bologna University was established by State act number 80 dated 14/2/1963 and subsequent modifications, combined with resolution number 68 of the University Assistance Board (Azienda Comunale per il Diritto allo Studio) of Bologna, passed on 17/5/1990.

The existing procedures are based on a set of programs, developed with traditional programming techniques, each treating a specific partial aspect of the study benefit law. These procedures can manage only standard cases correctly: particular cases are dealt with manually or by means of *ad hoc* programs.

With this prototype, our basic purpose was to build a system able to treat all cases expressly contemplated by existing regulations.

The often desirable isomorphism between legal text and logic program (cf. Routen 1988) could be achieved only to a limited degree, partly because the legal texts we used were not clearly formulated. We think that, for some aspects of the law on study benefits, our logic program gives a clearer representation, which may suggest some improvements to the natural language text.

STRUCTURE OF THE PROGRAM

The legal effects established by the study benefits law are the right to enrolment on the Assistance file of the University Assistance Board and to some economic benefits.

Enrolment on the Assistance File is dependent on some general requirements and certain economic, family and study conditions.

Economic and family conditions determine the student's economic category, while the student's scholastic curriculum determines the merit level. The combination of economic category and merit level establish the "layer" of beneficiaries the student is to be included in. Each layer entails a fixed set of benefits.

METHODOLOGY

The study benefits law allows an easy top-down structuring. In fact, the main rule of the program is the following:

```
has_right(Name,Surname,
                enrolment_in_Assistance_File(Layer)):-
     satisfies_general_requirements(Name,Surname),
     has_economic_category(Name,Surname,Category),
     has_merit_level(Name,Surname,Level),
     has_layer(Name,Surname,Level,Category,Layer).
```

The program is divided into four parts, each developing the definition of one of the predicates appearing in the body of the rule above. For example, these are the clauses defining the predicates 'satisfies_general_requirements' and 'has_economic_category'.

```
satisfies_general_requirements(Name,Surname):-
     attends_university_school(Name,Surname,
                                  UniversitySchool),
     satisfies_enrolment_requirements(Name,Surname),
     has_nationality(Name,Surname,Nationality),
     satisfies_nationality_requirements(Name,Surname,
                                  Nationality).

has_economic_category(Name,Surname,Category):-
     form('Economic Requirements',[Name,Surname]),
     family_income_of(Name,Surname,ValidIncome),
     kind_of_family_job_of(Name,Surname,KindJob),
     number_of_family_members_of(Name,Surname,
                                  NumFam),
     unemployed_children_in_the_family_of(Name,
                    Surname,UnemployedChildren),
     category(Name,Surname,KindJob,ValidIncome,
                                  Category).
```

The predicates appearing in the body of these clauses are defined by other clauses of the program (as satisfies_enrolment_requirements and satisfies_nationality_requirements) or are not defined, and are to be satisfied through querying the user (for example has_nationality and number_of_family_members_of).

Belonging to each layer entails a defined set of rights, as represented in the following clauses:

```
has_right(Name,Surname,Benefit):-
     has_right(Name,Surname,
               enrolment_in_Assistance_File(Layer)),
     guarantees(Layer,Benefit).

guarantees('A','Fixed grant for the studies').
guarantees('A','Accomodation and rent contribution').
guarantees('B','Accomodation and rent contribution').
...
```

We have deemed that the user would normally be interested in knowing all the benefits he has a right to.

So, the following rule has been defined, whose activation offers a complete consultation of the system:

```
benefits_evaluation_of(Name,Surname):-
   has_right_to(Name,Surname,
               enrolment_in_Assistance_File(Layer),
   print_benefits(Name,Surname,Category,Level,Layer).
```

The predicate print_benefits displays on the screen or writes to a file the set of the benefits the applicant is entitled to.

In our application the main theoretical problems concerning the legal applications of artificial intelligence are only marginally involved. Let us recall the following:

- Analogical reasoning. The study benefits law application does not involve many problems that have to be solved by analogical reasoning. There are some interpretation problems concerning concepts used but not defined in the legal text: for example the concepts of "valid degree", or of "member of the Italian nation". At present, the corresponding predicates are not defined in the program, and have to be solved querying the user. In the future, in relation to user requirements, we will consider whether those predicates can be defined or whether they must be treated differently.

- Deontic logic. Study benefits law assigns rights to the university students, rights that have been formalized in the structure:

```
has_right(Person,Content).
```

Nevertheless, no specific deontic inference procedure seems necessary.

- Non monotonic reasoning. Some default inferences involved in the application could be treated by means of negation by failure. Other problems were not considered in our prototype.

THE USE OF QUESTIONNAIRES

Questionnaires simplify the system-user interaction: instead of answering about twenty questions with twenty corresponding templates, the user has to fill in only five or six questionnaires. The system consultation becomes more friendly, and above all, the user can give most of the information concerning a certain problem on only one template, and so he can better understand the meaning of each question. Moreover, using questionnaires corresponds to the habits of most users, both the employees who are already using computer systems, and the students who, like all of us, are used to filling in forms when dealing with bureaucracy.

CONDITIONAL ANSWERS

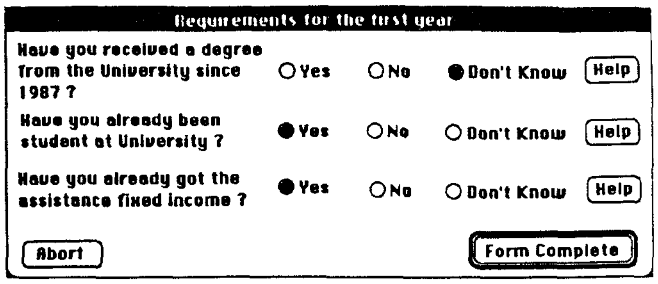
In our system, the possibility of obtaining conditional answers is very important, especially for direct use by students. If a student does not know some datum about his curriculum or his economic level, he can answer *Assume true* or *Don't know*. In this case, he can obtain not only a positive or negative answer, but also the indication of the conditions concerning the assumed predicates that have to be solved in order to obtain a positive answer.

The possibility of leaving some predicates undefined so as to obtain a conditional answer has a general interest in the legal domain. In fact, one of the principal criticisms of the application of law through computer-based systems is that these systems normally obtain the description of the facts of the case as an input from the user, and then establish the legal consequences. In this way a separation is created between the description (or qualification) of the facts and the application of the legal rules (cf. Bing 1989). These moments are, instead, strictly connected in the activity of the lawyer. In fact, the lawyer qualifies the facts of the case in consideration of the legal rules that can be applied to them, that is looking at the normative consequences deriving from that qualification. For example, let us consider the problem of qualifying the working activity of a student's family as independent (self employed) or as subordinate (employee). In a dubious case, the lawyer would bear in mind that stricter conditions are provided for independent work, so that possibly, for the same revenue level, less benefits would be granted if the family work were so qualified.

This separation between case description and legal consequence can be overcome by conditional answers. In fact, the user of our system is not obliged to answer all the questions he is asked. Problematic qualifications can be left undefined, so as to obtain a conditional answer indicating the conditions that have to be satisfied to obtain the legal consequence the query is about. The user can then decide how to complete the case description.

Figures 5 and 6 show an example of user-system interaction with conditional answers.

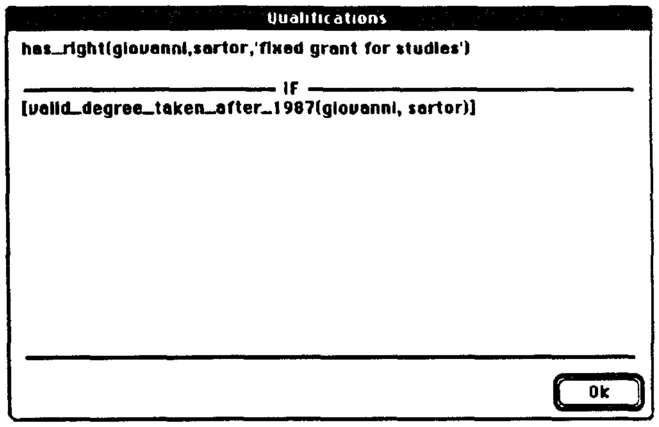
Figure 5: *'Don't know' answer by the user*



Requirements for the first year

Have you received a degree from the University since 1987 ?	○ Yes	○ No	● Don't Know	Help
Have you already been student at University ?	● Yes	○ No	○ Don't Know	Help
Have you already got the assistance fixed income ?	● Yes	○ No	○ Don't Know	Help

Figure 6: *Conditional answer by the system*



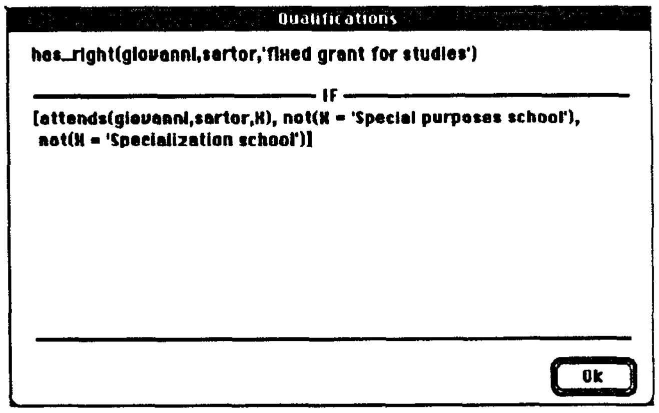
Qualifications

```
has_right(giovanni,sartor,'fixed grant for studies')

——————————————— IF ———————————————
[valid_degree_taken_after_1987(giovanni, sartor)]
```

Conditional answers can be generated by the rule of constructive negations. Let us assume that the applicant has declared not to be an Italian citizen and has answered *Don't know* to the menu of fig. 3 (choice of the school). If the remaining conditions for obtaining a grant are satisfied, the system will give a positive but conditioned (qualified) answer: the student has a right to the grant if he attends a school other than a special purposes or a specialization school.

Figure 7: *Conditional answer with constructive negation*



Qualifications

```
has_right(giovanni,sartor,'fixed grant for studies')

——————————————— IF ———————————————
[attends(giovanni,sartor,X), not(X = 'Special purposes school'),
 not(X = 'Specialization school')]
```

The qualification of the answer has been produced on the basis of the following clauses

```
satisfies_nationality_requirements(Name,Surname,
                                   Nationality,UniversitySchool):-
   not(nazionality(Name,Surname,'Italian')),
   not(post_university_school(UniversitySchool)),
   satisfies_foreign_nationality_requirements(Name,
                                        Surname,Nationality).

post_university_school('Special purposes school').
post_university_school('Specialization school').
```

The first states that non Italian students, in addition to the specific conditions concerning their own nationality, have to be enrolled in a school other than a post university school (only Italians are assisted to attend this kind of schools). The following assert that special purposes and specialization schools are post university schools.

FUTURE APPLICATIONS OF THE SYSTEM

We have in mind two types of users:

- students who have to decide whether to submit an application for study benefits, or have to prepare it, or want to check that their case has been correctly handled;
- public employees who have to apply the study benefits law.

We think that this prototype must be extended in different ways, to satisfy the requirements of these two types of users, although the core of the program, the logical representation of the study benefits law can be a single one.

- To meet the students' needs, the system should include additional facilities. In particular, if the conditions for registration in the University Assistance File are satisfied, the paper form for the application to the University Assistance Board should be automatically drafted, on the basis of the information given by the student during the consultation session. If the information given by the student is incomplete, only some slots of that form will be filled by the system. The conditional answer, printed separately, will give information to fill the remaining slots.

-To meet the requirements of the Assistance Board a connection with external data banks is necessary (for example, with the register of births, marriages, and deaths, and with the tax register), in order to check the correctness of the data contained in the application forms presented by the students. Furthermore, if a student's application is rejected, or when a student asks for explanations, a justification of the conclusion reached by the system, written in natural language, should be produced.

BIBLIOGRAPHY

Andretta, M., M. Lugaresi, F. Zambon, M. Losano, and N. Nannini. 1988. Uso di linguaggi formali per la rappresentazione di testi normativi: il progetto PROLEG (PROlog applicato alle LEGgi). In *Gulp 88. Atti del terzo convegno nazionale sulla programmazione logica*, 373-383.

Bing, J. 1990. *Three Generations of Computerized System for Public Administration and Some Implications for Legal Decision Making*. In *Ratio Juris* 3 (2): 219-236.

Chan, D. 1989. Constructive Negation Based on the Completed Database. In *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, edited by R.A. Kowalski, and K.A. Bowen, 111-125.

Cosmadopoulos, Y.A., and R.W. Southwick 1989. *Using Meta-Level Information for Expert System Control: A 'Blending' Transformer Approach*. Technical Report. London: Imperial College.

Routen, T. 1989. Hierarchically Organised Formalisations. In *The Second International Conference on Artificial Intelligence and Law. Proceedings of the Conference*, 242-250. New York: ACM.

Sergot, M.J. 1983. A Query-the-User Facility for Logic Programming, in *Integrated Interactive Computer Systems*, edited by P. Degano and E Sandewall, 27-41. Amsterdam: North Holland.

Sergot, M.J., and Y.A. Cosmadopoulos, 1991. *The Logic Programming System Skilaki: Design and Implementation*, Technical Report. London: Imperial College.

Sergot, M.J., and P. Hammond 1983. A PROLOG Shell for Logic Based Expert Systems. In *Expert System 83: Proceedings of the 3rd Technical Conference of the British Computer Society Specialist Group on Expert Systems* (Cambridge, December 1983), 94-104. British Computer Society.

Sergot, M.J., F, Sadri, R.A. Kowalski, F. Kriwaczek, P. Hammond, and H.T. Cory. 1986. The British Nationality Act as a Logic Program. *Communications of the ACM*. 29: 370-386.

Wolstenhome, D.E. 1987. Saying "I don't know" and conditional answers. In D. S. Moralee, editor, *Research and Development in Expert Systems IV*, pages 115-125, Cambridge University Press.