Artificial Intelligence and The Law:

How to Develop a Rule-Based Legal Expert System in Prolog

By Francisco Eymael Garcia Scherer

Introduction

This article demonstrates the key techniques for developing a rule-based legal expert system. It also introduces the reader to basic aspects of PROLOG, such as goal-oriented programming, backtracking and the interaction between PROLOG facts and variables.

This article first analyzes the formalization of legal rules and the development of a logic based decision system (i.e., an algorithm). It then illustrates how to develop a legal expert system in PROLOG (historically a key language for the development of artificial intelligence systems) based on the decision system algorithm, described above.

Formalization of Legal Rules and the Development of a Logic Based Decision System

We first review the formalization of a simple sentence followed by the formalization of compounded sentences. Next we look at a logic-based decision system.

The Formalization of Simples Sentences

According to the canons of propositional logic, a capital letter can symbolize any simple English sentence. A simple sentence, for the purposes of this discussion, is any English sentence composed of a simple subject and a predicate. For example: “Laura is a good student”, “Mary bought a new radio”, etc. We can symbolize the sentence “John has a TV set” with the capital letter *J*. Considered a logical variable, this capital letter assumes one of two logical values: one or zero. If considered true, then its J’s logical value is one; otherwise, it is false with a logical value of zero.

The Formalization of Compounded Sentences

According to the canons of propositional logic, there are six logical operators: *NOT*, *AND*, *OR*, *XOR*, the *Conditional Operator* and the *Biconditional Operator*. These operators are, respectively, represented by the symbols ~, ·, ∨, ^. ⊃ and ≡ . The sentence “John has not a TV set”, is represented by ~*J*. Truth Table 1 defines the truth-value of this compounded sentence.

J	~J
1	0
0	1

Table 1: Truth table for the algebraic expression ~J

A compounded sentence, for the purposes of this discussion, is any English sentence composed of at least one simple sentence and the logical operator NOT or any English sentence composed by at least two simple sentences coordinated by a logical operator. The sentence "Laura is a good student and Mary likes to play tennis", for instance, can be formalized as L•M, where L stands for "Laura is a good student", M stands for "Mary likes to play tennis" and • stands for the logical operator AND. The truth value of the algebraic expression L•M is determined according to Truth Table 2.

L	M	L · M
1	1	1
0	1	0
1	0	0
0	0	0

Table 2: Truth table for the algebraic expression L · M.

By the same token, the complex sentence “John is 13 years old or John is mentally ill” can be represented by the algebraic expression O ∨ M, where O stands for “John is 13 years old”, the symbol ∨ stands for the logic operator

OR and M stands for “John is mentally ill”. Table 3 defines the truth-value of this compounded sentence.

O	M	O v M
1	1	1
0	1	1
1	0	1
0	0	0

Table 3: Truth table for O v M.

It is important to understand that the conjunction OR can only be represented by the symbol ∨ in the formalization of coordinated sentences that are not mutually exclusive. It is perfectly possible that “John is 13 years old” AND that “John is mentally ill”. On the other hand, the sentence “Either George Bush Senior or Bill Clinton was the President of the US in 1994” cannot be formalized with the logical operator ∨, since both hypothesis are mutually exclusive. This sentence can only be formalized as G ^ B, where G stands for “George Bush Senior was the President of the US in 1994”, B stands for “Bill Clinton was the President of the US in 1994” and the symbol ^ stands for the logical operator XOR. Table 4 defines the truth-value of this compounded sentence.

G	B	G ^ B
1	1	0
0	1	1
1	0	1
0	0	0

Table 4: Truth Table for G ^ B.

The sentence “If Mary is a minor, then Mary is legally incompetent”, can be symbolized by the algebraic expression M ⊃ L, where M stands for “Mary is a minor”, the symbol ⊃ stands for the conditional operator “if-then” and L stands for “Mary is legally incompetent”. In this conditional algebraic expression, the variable M is the “antecedent”, while the variable L is the “con-sequent”. Table 5 defines the truth-value of this compounded sentence.

M	L	M É L
1	1	1
0	1	1
1	0	0
0	0	1

Table 5: Truth Table for M É L.

Finally, the sentence “Christine will be allowed to travel with her friends if and only if she passes the final exam on English” has the algebraic expression C ≡ P. Here C stands for “Christine will be allowed to travel with her friends”, ≡ stands for the biconditional operator “if and only if” and P stands for “Christine passed the final exam on English”. In this biconditional algebraic expression, the variable P is the antecedent of the algebraic expression, while the variable C is the consequent. Truth Table 6 defines the truth-value of this compounded sentence.

C	P	C ° P
1	1	1
0	1	0
1	0	0
0	0	1

Table 6: Truth Table for C ° P.

The above statements are “truth-functional”, i.e., the determination of the logical value of these compounded sentences rests solely on the truth-value of the simple sentences from which they are composed. There are, however, “non-truth-functional” compounded sentences, i.e, compounded sentences where the logical value is not determined solely by the truth-value of the simple sentences that compose them. These sentences are coordinated or joined by “non-truth-functional” operators such as “X believes that ....” (X being any person), “It is possible that ....”, “It is necessary that ....”, “....because....”, “....after....”. In general, all of these non-truth-functional operators relate to concepts of time, possibility and belief. According to the canons of propositional logic, we symbolize these compounded sentences with single capital letters. For instance, the sentence “Christine quit her job because she did not like her boss” is symbolized as C.

The Development of a Logic-Based System of Decision

According to the Occupations Code of the State of Texas, in its Section 2501.001, the term “Personnel Service” means “a person who, regardless of whether for a fee, directly or indirectly offers or attempts to obtain permanent employment for an applicant or obtains

or attempts to obtain a permanent employee for an employer (...)".

Section 2501.051, states that: "A person may not own a personnel service that operates in this state unless the person holds a certificate of authority issued under this chapter".

And Section 2501.055, finally, declares that:

"(a) On receipt of a notice filed under Section 2501.053, the commissioner shall issue to the owner a certificate of authority to do business as a personnel service not later than the 15th day after the date the notice is filed if the owner:

(1) pays the filing fee required for the certificate; and
(2) complies with the requirements of Section 2501.054.

(b) A certificate of authority is valid for the period set by the Texas Commission of Licensing and Regulation."

In this subsection of the article, we develop a system of analysis and decision that enables us to analyze any notice filed under Section 2501.053 and determine whether the correspondent certificate of authority should or should not be issued by the commissioner mentioned in the Section 2501.055.

And section 2501.053 states that:

(a) Not later than the 30th day before the date a personnel service begins operating in this state, the owner of the service must file notice with the commissioner.
(b) The notice must include:

(1) the address of each location at which the personnel service is to operate its business on a daily basis;
(2) the assumed name, if any, under which the personnel service will operate;
(3) the name and residence address of each owner; and
(4) a statement that each owner has read and is familiar with this chapter.

(c) The notice must be signed and sworn to by the owner before a notary public or other officer authorized to administer oaths.

The Section 2501.053 can be interpreted and formalized as follows:

$$L \equiv (F \cdot A \cdot N \cdot O \cdot R \cdot S)$$

By definition, the Truth-Value (TV) of this algebraic expression is one. Table 7 defines the meaning and truth-value of the Variables (V) for this algebraic expression. *U* stands for unknown.

V	Meaning	TV
L	The notice under analysis was filled according to the Section 2501.053	u
F	The owner of the service filed the notice with the commissioner not later than the 30th day before the date on which the personnel service will begin to operate in this state.	u
A	The notice includes the address of each location at which the personnel service is to operate its business on a daily basis.	u
N	The notice includes the assumed name, if any, under which the personnel service will operate.	u
O	The notice includes the name and the address of each owner.	u
R	The notice includes a statement that each owner has read and is familiar with this chapter.	u
S	The notice was signed and sworn to by the owner(s) before a notary public or other officer authorized to administer oaths.	u

Table 7: Meaning and truth value of logical variables.

Section 2501.054, states that:

"(a) An owner who files a notice under Section 2501.053 shall file with the notice a bond in the amount of $5,000 that is:

(1) executed with a good and sufficient surety;
(2) payable to the state; and
(3) conditioned that the obligor will not violate this chapter.

(b) In lieu of a bond under Subsection (a), the owner may deposit $5,000 in cash.

(c) A bond filed under Subsection (a) must state that a person aggrieved by a violation of this chapter by the principal or an agent or representative of the principal is entitled to bring an action on the bond.
(d) An owner of a personnel service may satisfy the requirements of this section by filing one bond for the personnel service, regardless of the number of locations at which the personnel service is to operate its business on a daily basis."

The section 2501.054 can be interpreted and formalized as:

$$Q \equiv B1 \bullet (B2 \bullet B3 \bullet B4 \bullet (L1 \wedge (\sim L1)) \bullet V \bullet (D \wedge (\sim D)))$$

By definition, the Truth-Value of this algebraic expression is one. Table 8 (page 37) defines the meaning and truth-value of the variables for this algebraic expression.

Finally, the Section 2501.055 can be interpreted and formalized as:

$$(C \bullet Y \bullet Q) \supset I$$

As in the earlier cases, we define the truth-value of this algebraic expression to be one (true). Truth Table 9 defines the meaning and truth-value of the Variables for this algebraic expression.

In addition to this rule of conduct, it is also possible to define a second rule of conduct - based on the Section under analysis. We could formalize it as:

$$(\sim C) \vee (\sim Y) \vee (\sim Q) \supset (R1 \bullet (\sim I1))$$

Again, we define the truth-value of this algebraic expression as one. Table 10 defines the meaning and truth-value of R1 and I1.

Finally, we can formalize subsection (b) of Section 2501.055 as a simple sentence, V2, which truth value is defined as one.

V	Meaning	TV
Q	The owner of the personnel service complied with the requirements of the section 2501.054	u
B1	The owner who filed the notice under Section 2501.053 filed with the notice a bond in the amount of $ 5.000.	u
B2	The bond in question is executed with a good and sufficient surety.	u
B3	The bond in question is payable to the state.	u
B4	The bond in question is conditioned that the obligator will not violate this chapter.	u
L1	The owner deposited $ 5.000 in cash in lieu of the bond.	u
V	The bond in question states that a person aggrieved by a violation of this chapter by the principal or an agent or representative of the principal is entitled to bring an action on the bond.	u
D	The owner of the personnel service filed one bond for each location at which the personnel service is to operate its business on a daily basis	u

Table 8: Meaning and truth value of logical variables.

In table 11, we summarize the set of algebraic expressions described in the previous paragraphs. Although the algebraic expression I has not yet been described, it will be employed in the decision system that is described next. So, it was included in table 11 as algebraic expression E7. This also applies to the algebraic expression C ≡ L, which was included in table 11 as algebraic expression E2.

It is possible to interpret each line of this table as a node of the decision system. The system will analyze the truth-value of the algebraic expression (and its respective variables) related to the node one (N1).

Since we previously defined the truth-value of E1 as one (true), the system initiates the analysis of node N2 and other nodes until node N6. At this point, the system analyzes the truth-value of E7 (which corresponds to the variable I) and if it is one, then the system starts the analysis of N7. On the other hand, if the truth-value of E7 is unknown, then the system simply finishes the execution of this algorithm.

The Development of a Legal Expert System in Prolog

The remainder of this article covers Prolog facts, Horn Clauses, Backtracking, Recursion, the Development of a Legal Expert System and a brief look at the Prolog Code.

Prolog Facts

A Prolog Fact is a statement that has been registered in Prolog code. The statement "John is a good soccer player", for instance, can be coded in Prolog as:

john_is_a_good_soccer_player.

The first letter of a Prolog fact must be in lower-case. Otherwise, the Prolog listener (part of the Prolog System) will interpret the statement as a variable.

After registering this statement in a Prolog database, we will later be able to recall this same information through a query. The Prolog Listener loads the database and allows the entry of the above statement.

The Listener interprets this statement as a question and it searches the consulted database attempting to find a fact that matches the query. If this fact is found, then the Listener answers "yes", as it will in this case. Otherwise, the system answers "no", which indicates that the query found no match. In other words, a query is a goal, which either succeeds (with the answer "yes") or fails (with the answer "no").

There are two types of Prolog facts: simple and complex. The statement above is an example of a simple Prolog fact. A complex Prolog fact is composed of a *functor* and at least one *argument*. The statement "John is a good soccer player", can be formalized as:

is_a_good_soccer_player(john).

In this case, the functor is the term "is_a_good_soccer_player" and the argument is "john". To determine who is a good soccer player, the query would be:

is_a_good_soccer_player(X).

The argument is replaced by a variable, the first letter of which is always an upper case letter. The listener searches the database for a fact that matches the query, and the variable X is instantiated to the argument of the fact registered in the database. In this case, it will produce the following answer:

X = john.

The system then waits for the next command. When we press *ENTER*, the system finishes the query routine and generates the answer "yes". If we press ";", the system ignores the first match and reinitiates the search for another match.

V	Meaning	TV
C	The commissioner received a notice filed according to the Section 2501.053.	L
Y	The owner paid the filing fee required for the certificate.	u
Q	The owner complied with the requirements of Section 2501.054	u
I	The commissioner shall issue to the owner a certificate of authority to do business as a personnel service not later than the 15th day after the date the notice is filed.	u

Table 9: Meaning and truth value of logical variables.

V	Meaning	TV
R1	The commissioner shall reject the notice under analysis.	u
I1	The commissioner shall issue the certificate of authority required by the owner of the personnel service in question.	u

Table 10: Meaning and truth value of logical variables.

En	Algebraic Expression
E1	L ≡ (F • A • N • O • R • S)
E2	C ≡ L
E3	Q ≡ B1 • (B2 • B3 • B4 • (L1 ^ (~ L1)) • V • (D ^ (~D)))
E4	(C • Y • Q) ⊃ I
E5	(~ C) ∨ (~ Y) ∨ (~ Q) ⊃ (R1 • (~I1))
E6	V2
E7	I

Table 11: A summary of the set of algebraic expressions described in the previous paragraphs.

Nn	Algebraic Expression	En = 1	En = 0	En = U
N1	E1	N2	-	-
N2	E2	N3	-	-
N3	E3	N4	-	-
N4	E4	N4	-	-
N5	E5	N6	-	-
N6	E7	N7	-	End
N7	E6	End	-	-

Table 12: A system of decision for the legal expert system.

In this case, the search fails and the system answers "no".

We can formalize a Boolean algebraic expression as a complex Prolog fact. The algebraic expression A&B, for instance, can be formalized as &(a,b), in which the functor & represents the logical operator AND, with arguments 1 and 2 representing, the variables A and B, respectively. This is not, however, a very practical way of codifying an algebraic expression.

Through the command (or *predicate*) op/3, we can declare that the term "&" is an *operator*, allowing the programmer to register the algebraic expression above in the conventional manner (see Listing 1). We must register this statement at the top of the database, so the Listener is able to acknowledge this statement and apply it to the algebraic expressions registered in the lines of code that follow.

This declaration, in Prolog, is coded as:

```
:-op(597,xfy,&).
```

The remaining logical operators are defined in the same manner, allowing the programmer to register any algebraic expression in the conventional way.

The algebraic expression E1, for instance, is coded as follows:

```
% Define the logical operator NOT
:-op(590,fy,neg).
% Define the logical operator AND
:-op(597,xfy,&).
% Define the selective logical operator AND
:-op(596,xfy,@).
% Define the logical operator OR
:-op(599,xfy,#).
% Define the logical operator XOR
:-op(598,xfy,<>).
% Define the biconditional logical operator
%      (If and Only If)
:-op(700,xfy,<=>).
% Define the conditional logical operator
%      (If - Then)
:-op(650,xfy,=>).

% These definitions are essential for the
%       stand-alone version of the program
:- discontiguous exp/3.
:- discontiguous analise/3.
:- discontiguous sign/4.
:- discontiguous fund/3.
:- dynamic valor/4.
:- discontiguous node/2.
```

Listing 1: Operators and Definitions

```
% Legal Rules
exp(a1,e1,l<=>(f&a&(n<>neg(n))&o&r&s)).
sign(a1,e1,l,'The notice under analysis was filled according to the Section 2501.053 of the
   Occupations Code of the State of Texas').
sign(a1,e1,f,'The owner of the service filed the notice with the commissioner not later than
   the 30th day before the date on which the personnel service will begin to operate in the
   State of Texas').
sign(a1,e1,a,'The notice includes the address of each location at which the personnel
   service is to operate its business on daily basis').
sign(a1,e1,n,'The notice includes the assumed name, if any, under which the personnel
   service will operate').
sign(a1,e1,o,'The notice includes the name and the address of each owner').
sign(a1,e1,r,'The notice includes a statement that each owner has read and is familiar
   with this chapter').
sign(a1,e1,s,'The notice was signed and sworn to by the owner(s) before a notary public
   or other officer authorized to administer oaths').
valor(a1,e1,l<=>(f&a&(n<>neg(n))&o&r&s),truth).

exp(a1,e2,c<=>l).
valor(a1,e2,c<=>l,truth).
sign(a1,e2,c,'The commissioner received a notice filed according to the Section 2501.053
   of the Occupations Code of the State of Texas').
```

Listing 2: Sample of Legal Rules

```
exp(a1,e1,l<=>(f&a&n&o&r&s)).
```

As a statement, this Prolog fact declares that the algebraic expression corresponds to the algebraic expression E1 of the decision system A1. The meaning of variable L can be coded as:

```
sign(a1,e1,l," The notice under analysis was filled according to the Section 2501.053.").
```

And, finally, the truth-value of the algebraic expression E1 can be registered as:

```
valor(a1,e1,l<=>
(f&a&n&o&r&s),truth).
```

Horn Clauses

From a procedural view-point, a horn clause is the Prolog equivalent of the classic *IF – THEN* command of conventional languages such as BASIC. It is composed of a *head of clause*, the operator ":-" and the *body of the clause*. The head of the clause is the antecedent of the horn clause, while the body is the consequent of the head of the clause.

The following code line, for instance, in an example of a horn clause:

```
a:-b.
```

This declares that if the goal is *a*, then the system must search for *b*. That is, the goal *a* will succeed if the system finds a match for the goal *b*. Otherwise, it will fail. Thus *b* can be defined as a subgoal of the main goal *a*.

The body of the clause can be composed of multiple goals, which can be either sequential or alternative in nature. The body of the following horn clause, for instance, is composed of a sequence of subgoals:

```
a:-b,c,d.
```

The code line in question is interpreted as "If the main goal is "a", then execute the goal "b". If b succeeds, then execute the goal c. If c succeeds, then execute the goal d. The goal *a* only succeeds if and only if the goals b, c and d succeed.

The body of the following horn clause is composed of alternative subgoals:

```
a:-b;c;d.
```

This code line is interpreted as "IF the main goal is *a* THEN execute the goal *b*. If *b* succeeds, then *a* also succeeded. Otherwise, execute *c*. If *c* succeeded, then *a* also succeeded. Otherwise, execute *d*. If *d* succeeded, then *a* also succeeded. Otherwise, *a* fails.

Backtracking

Let us look at the following lines of code:

```
a:-b,c.

b:-b1;b2.

b1.

b2.
```

Let us assume that our main goal is *a*. In this case the subgoal *b* will suceed and the subgoal *c* will fail. The system, then, will *backtrack* and retest *b*. The first solution for *b* was *b1*. The system will search for a new solution for *b*, which is *b2*. The system will then test *c* again. Since *c* was not declared as a Prolog fact, this search fails. The listener will backtrack again and search for a third solution for b. This search fails, as there are no other solutions for this subgoal. Only after checking all of these options will the Listener answer *no*, indicating that the main goal *a* failed.

Backtracking is one of the most useful and tricky features of Prolog, and the programmer must learn to control it to develop a functional software in this language. We can use the predicate once/1 since it establishes that a given goal cannot be tested again in case of a failure followed by backtracking. On reviewing the following code:

```
a:-once(b),c.
```

the failure of *c* automatically results in the failure of *a*, since the subgoal *b* would not be tested again.

In our legal expert system shell, we use the predicate once/1 as follows:

```
node(A,Q,W,S,N):-
      once(truth(A,Q,W)),node(A,S);
      once(false(A,Q,W)),node(A,N).
```

Recursion

A recursive clause is a horn clause that contains at least one subgoal that is identical to the head of the clause. See the following horn clause:

```
truth(A,Q,X):-
      atom(X),valor(A,Q,X,truth);
            functor(X,&,2),
               valor(A,Q,X,truth);
            functor(X,&,2),
               arg(1,X,B),arg(2,X,C),
               once(truth(A,Q,B)),
               truth(A,Q,C);
```

```
% Main

main:-nl,write('IN ORDER TO ANSWER THE QUESTIONS
    FORMULATED BY THE SYSTEM, SIMPLY
PRESS T (true) OR F (false) AND PRESS <ENTER>.'),nl,node(a1,n1),
    nl,write('Press Any Key to Finalize the Execution of the Program'),nl,read_string(Q).

% Algorithm
node(a1,n1):-node(a1,e1,n2,_).
node(a1,n2):-node(a1,e2,n3,_).
node(a1,n3):-node(a1,e3,n4,_).
node(a1,n4):-node(a1,e4,n5,_).
node(a1,n5):-node(a1,e5,n6,_).
node(a1,n6):-metanode(a1,e7,n7,_,end).
node(a1,n7):-node(a1,e6,end,_).

% The Node System
node(A,Q,S,N):-
            exp(A,Q,W),once(analise(A,Q,W)),node(A,Q,W,S,N).
node(A,Q,W,S,N):-
             once(truth(A,Q,W)),ncond(A,Q,W),node(A,S);
             once(false(A,Q,W)),ncond(A,Q,W),node(A,N).
node(_,end).

% The predicate ncond/3 will generate an output based on the truth value of non-conditional
% algebraic expressions.  This predicate always suceeds
ncond(A,Q,W):-
not(functor(W,=>,2)),not(functor(W,<=>,2)),print(A,Q,W),justify(A,Q,W);
             true.
```

Listing 3: Main program and node system.

```
% The predicate justify/3 indicates the legal basis of a given output generated by the system

jJustify(A,Q,W):-
            functor(W,=>,2),arg(2,W,Y),functor(Y,=>,2);
            functor(W,=>,2),arg(2,W,Y),functor(Y,<=>,2);
            functor(W,<=>,2),arg(1,W,X),functor(X,=>,2);
            functor(W,<=>,2),arg(1,W,X),functor(X,<=>,2);
            fund(A,Q,O),nl,write('LEGAL BASIS :  '),nl,write(O),nl;
            true.

% input/3 queries the user about the truth value of X
input(A,Q,X):-once(sign(A,_,X,W)),nl,write('DEFINE THE FOLLOWING STATEMENT AS TRUE
    OR FALSE : '),nl,nl,
            write(W),write('.'),nl,nl,write('ANSWER : '),read_string(P),input2(A,Q,X,P);
            true.
input2(A,Q,X,W):-W=='T',asserta(valor(A,_,X,truth)).
input2(A,Q,X,W):-W=='F',asserta(valor(A,_,X,false)).
input2(A,Q,X,W):-nl,write('The System Will not Accept this Answer'),nl,input(A,Q,X).

% definet/3 defines the truth value of Y as 1
% If the algebraic expression is coordinated by either "-" or "#", then the algebraic expression
% as whole is defined as 1 and the truth value of its respective variables is not defined in
% the database

definet(A,Q,Y):-valor(A,_,Y,truth);
            valor(A,_,Y,false),write('CONTRADICTION '),write(Y);
            atom(Y),asserta(valor(A,_,Y,truth)).
definet(A,Q,Y):-functor(Y,neg,1),arg(1,Y,X),definef(A,Q,X).
definet(A,Q,Y):-functor(Y,&,2),arg(1,Y,X),definet(A,Q,X),arg(2,Y,Z),definet(A,Q,Z),nl.
definet(A,Q,Y):-functor(Y,@,2),arg(1,Y,X),definet(A,Q,X),arg(2,Y,Z),definet(A,Q,Z),nl.
definet(A,Q,Y):-functor(Y,#,2),asserta(valor(A,Q,Y,truth));
            functor(Y,<>,2),asserta(valor(A,Q,Y,truth));
            functor(Y,=>,2),asserta(valor(A,Q,Y,truth)),analise(A,Q,Y);
            functor(Y,<=>,2),asserta(valor(A,Q,Y,truth)),analise(A,Q,Y).
```

Listing 4: The predicates justify/3, input/3, input 2/4 and definet/3.

This clause declares that a given algebraic expression X (which corresponds to the algebraic expression Q of the decision system A) is considered true if X is a single variable (i.e., an "atom") and if the truth-value of X has been defined as true in the database. Alternatively, X will be considered true if X is an algebraic expression coordinated by the operator AND (i.e., "&") and if the truth-value of X has been defined as true in the database. Finally, X will also be considered true if X is an algebraic expression coordinated by the operator AND and if both coordinated algebraic expressions (i.e., "A" and "B") are considered true.

The Development of a Legal Expert System

Based on the theoretical concepts above, it was possible to develop the predicates in Table 13. A Prolog predicate corresponds to a simple or complex Prolog fact, which may or may not constitute the head of a horn clause. For the purposes of this article, we will assume that there are two types of Prolog predicates: Built-In predicates and Non-Built-In predicates. The first type of predicate is already defined in the structure of the language and can be used along with the Non-Built-in predicates (Examples of Built-in predicates: not/1, write/1, functor/3, etc). The user of the system defines the second type of predicate. The predicates listed in this subsection can be considered Non-Built-in predicates.

The predicate main/0 is the initial goal of our legal expert system and it invokes the first node of the decision system. After the legal expert system finishes execution of the algorithm, the predicate main/0 finishes the execution of the Prolog code.

The formalized system of decision is represented by a set of horn clauses. The head of each clause is a predicate node/2, while the body of each clause is a predicate node/4. Predicate node/4 invokes the predicate analise/3, which invokes known/3, to determine whether the truth-value of the algebraic expression indicated by node/4 can or cannot be deduced based on the information already registered in the database. If the truth-value cannot be determined based on the information already registered in the database, the predicate analise/3 invokes input/2 (which invokes input2/3), to query the user concerning the truth-value of the algebraic expression under analysis.

If the algebraic expression under analysis is a conditional algebraic expression, analise/3, based on the truth-value of the antecedent, also invokes either definet/3 or definef/3 to register the truth-value of the consequent. In this case, analise/3 also invokes conclusion/3, print/3 and justify/3. The predicate conclusion/3 prints the word "CONCLUSION". The predicate print/3 prints the verbal meaning of the consequent, and, finally, justify/3 indicates the legal basis of the conclusion generated by the system.

After the analysis of the algebraic expression indicated by node/2, the predicate node/4 invokes the predicate node/5, which, based on the truth value of the algebraic expression indicated by node/4, determines whether to call the node(A,S) or the node(A,N) (The node(A,S) is invoked if the truth value of algebraic expression is one and the

n	Predicates	n	Predicate
1	exp/3	13	truth/3
2	sign/4	14	false/3
3	valor/4	15	conclusion/3
4	fund/3	16	print/3
5	main/0	17	imppos/3
6	node/2	18	impneg/3
7	node/4	19	justify/3
8	node/5	20	input/3
9	metanode/5	21	input2/3
10	analise/3	22	definet/3
11	ncond/3	23	definef/3
12	known/3		

Table 13: Legal Expert System Predicates.

```
% The predicate conclusion/3 determines whether the expression "conclusion" shall be printer/3.
conclusion(A,Q,Y):-
        functor(Y,=>,2);
        functor(Y,<=>,2);
        nl,write('CONCLUSION : ');
        true.
% print/3 determines whether the truth value of W is 1 or 0 and invokes either imppos/3 or impneg/3
print(A,Q,W):-
        truth(A,Q,W),imppos(A,Q,W);
        false(A,Q,W),impneg(A,Q,W);
        true.
```

Listing 4: Predicates conclusion/3 and Print/3.

node(A,N) is invoked if the truth value of the algebraic expression is zero). Before invoking one of the above nodes, however, node/5 will invoke ncond/3, to check whether the algebraic expression indicated by node/4 is a non-conditional algebraic expression. In the case where the result is positive, ncond/3 invokes print/3 to print the verbal meaning of the algebraic expression, which also invokes justify/3 to indicate the legal source of the algebraic expression. Otherwise, ncond/3 simply succeeds and the next node/2 is invoked by node/5.

As for the predicate metanode/5, this predicate determines, based solely on information already registered in the database, the next node/2 to be called.

It is important to point out that known/3 is directly linked to truth/3 and false/3, which means that known/3 will succeed if and only if the algebraic expression under analysis is either true or false. It is also important to point out that, for procedural reasons, we have introduced a "selective" AND operator, which truth value is considered false if the truth value of the first coordinated algebraic expression (be it a single variable or a more complex algebraic expression) is defined as false by the user. The main objective of this operator is to avoid the formulation of senseless questions by the system, in some cases. Please consult the complete code for this article.

The Prolog Code

You can access the complete code for this article at the PC AI website.

Conclusion

In this brief article, we have demonstrated how to develop a simple legal expert system in Prolog. Due to the brief nature of the article, the subject could not be completely explored. A second article could be written just to review the elaboration of a routine for logical justification. In addition, the formulation of logical conclusions, based on incomplete information, could be explored in a completely different article. In spite of its conciseness, however, this article has shown that, based on simple principles of logic and computer programming, is it is possible to develop legal expert systems to be used as auxiliary tools in the analysis of concrete legal cases.

Reference

KLENK, Virginia. Understanding Symbolic Logic. 3rd Edition. Editor: Prentice Hall, Englewood Cliffs, New Jersey, USA.

BRATKO, Ivan. Prolog: Programming for Artificial Intelligence. 2nd Edition. Editor: Addison-Wesley, Edinburgh Gate, Harlow, United Kingdom.

AMZI! Prolog User's Guide and Reference.

The Occupations Code of the State of Texas, USA.

Francisco E. G. Scherer was born and raised in Porto Alegre, Brazil. He graduated from law school in 2003 and is currently working at CEF, a state-owned bank in Brazil. He can be reached at *eymael@yahoo.com* or *fancisco.scherer@caixa.gov.br*.