An Expert System Shell for Uncertain Rule- and Model-Based Reasoning
A. Niederliński
Department of Automation, Computer Science and Electronics, Silesian University of Technology,
Akademicka 16, 44-100 Gliwice
E-mail: aniederlinski@ia.polsl.gliwice.pl

Abstract
Detailed design assumptions and functional principles of two hybrid rule-and model-based expert system shells for uncertain backward and forward reasoning are presented. The systems may be used to reason with any knowledge base, which consists of following parts:

- Uncertain Rule Base, containing uncertain IF...THEN... rules used for reasoning
- Uncertain Model Base, containing exact arithmetic and uncertain relational models
- Exact Constraint Base, containing sets of exact mutually exclusive and completely exhaustive askable conditions
- Uncertain Constraint Base, containing sets of uncertain mutually exclusive and exhaustive askable conditions
- Advice Base, being a directory of advice corresponding to rules of the rule base.
- Advice Files, being text files with advice.

The Knowledge Base semantics is simple, intuitive and straightforward. Practically unlimited nesting of rules, models as well as rules and models is allowed. Uncertainty is modelled using a modified Stanford Certainty Factor Algebra. The classic Stanford Certainty Factor Algebra using Certainty Factors in the range [-1,1] has been modified to make it conform to Aristotelian Logic for the case of CF’s being equal 1 or –1 and take into account dependent and independent lists of conditions. A unified string representation for conclusions, conditions, arguments, certainty factors and results contributes to the system flexibility and easy of programming. No attribute structure is defined for conditions of the rule base. Instead mutually exclusive attribute values are defined using an Exact Constraint Base and Uncertain Constraint Base. The systems are equipped with diagnostic facilities automatically checking the rule- and constraint bases for inconsistencies and redundancies, providing warnings and detailed diagnostic messages. A Reasoning Report and Diagnostic Report are produced on the fly. They may be saved for analysis and record keeping.

Keywords , expert system shells; uncertain reasoning; certainty factor algebra

1. Background
The origin of this paper is educational. My first attempts at developing various expert system shells are dated almost 15 years ago, in order to provide students with tools that might be used to formulate decision problems as knowledge bases. The shells available at that time – of rather restricted nature – were in executable forms, which made their development and adjustment impossible. The only alternative was to write experts system shells from scratch, using some high level language like LISP [8], OPS, CLIPS [5],[6] or Prolog [4],[18]. The choice of Prolog was pretty obvious: it was the only declarative general-purpose language with a program structure closely resembling the structure of expert systems while at the same time providing without any programming efforts such useful services like backtracking and unification. The decision was supported by the existence of a very efficient and inexpensive Prolog compiler (Turbo-Prolog, to be developed latter into PDC Prolog and Visual Prolog).

The paper describes a possible solution to the problem, implemented in Prolog, see [10], [11]. Unlike traditional procedural languages (Pascal, C, C++ etc.), Prolog is a declarative language. The Prolog programmer does not build algorithms and does not specify how a solution is to be derived; instead he/she declares the available knowledge about the problem domain, leaving the rest to the inference mechanism build into Prolog compiler/interpreters. Therefore Prolog seems to be ideal for knowledge representation and reasoning tasks, see [7], [9], [12], [14], [18]. The systems discussed have been designed using PDC 331A PROLOG and are available as .exe files (Polish and English versions) on the website: http://www.ekspert.wsi.edu.pl/.

The solution is a major extension of the expert systems discussed in [9].

2. General design assumptions
The formulation of design assumptions for an Expert System Shell ([11],[3],[17]) amounts really to answering the question: What mechanisms and entities (rules, operations, constraints, recommendations) might be needed for modeling a relatively large and varied set of uncertain decision problems by stating a number of uncertain IF...THEN... rules, supported by arithmetic, logical and relational operations? The number of different mechanisms and entities should be as small as possible without however sacrificing their universality. The process of model formulation should be as free as possible from programming technicalities so as not to discourage domain experts from using it. The models created using these mechanisms should be the feedstock for appropriate expert system shells, testing them in
search of certainty factors for rule/model conclusions and/or numerical values of model outputs.

The Rule- and Model-Based Expert System Shell (RMES, [10], [11]) for uncertain reasoning is based upon the following design assumptions:

1. The knowledge base consists of following parts:
   - Uncertain Rule Base, containing uncertain IF...THEN... rules using Certainty Factors being defined by a modified Stanford Certainty Factor Algebra
   - Uncertain Model Base, containing exact arithmetic models and uncertain relational models
   - Exact Constraint Base, containing lists of exact mutually exclusive and exhaustive askable conditions
   - Uncertain Constraint Base, containing lists of uncertain mutually exclusive and exhaustive askable conditions
   - Advice Base, being a directory of advice files corresponding to rules of the rule base.
   - Advice Files, being text files with advice.
   - Abstracts, being text files summarizing the domain knowledge used to build the knowledge base.
2. Practically unlimited nesting of rules, of models and of rules and models is possible.
3. Forward Chaining and Backward Chaining is supported.
4. Rule Base and Constraint Base verification is performed automatically before reasoning begins.
5. Reasoning or Diagnostic Reports are produced on the fly and may be saved for analysis and documentation.

In the following the term “string variable” will be used to denote an entity of the following form: “Name” or “Value”, where ‘Name’ is any sequence of characters including spaces and ‘Value’ is a real number. Three types of string variables are used:

1. Logical string variables, for which ‘Name’ denotes a logical variable, being either true or false. E.g. “Good financial standing” is a logical string variable. “Name” = “no condition” denotes an logical string variable instantiated as always true; “Unit price” is an instantiated real string variable and “53.25” is an instantiated real string variable.
2. Integer, string variables, for which ‘Name’ denotes an integer variable or ‘Value’ denotes an integer number. E.g. “Client Number” is an uninstantiated integer string variable and “1053” is an instantiated integer string variable.
3. Real string variables, for which ‘Name’ denotes a real variable or ‘Value’ denotes a real number. E.g. “Unit price” is an instantaneous real string variable and “53.25” is a real string variable.

The justification for introducing this concept is the flexibility it offers with respect to e.g. using the same model as either a relational model with logical output or arithmetical model with real output, as well as using them in different situations, e.g. with some variables instantiated and others uninstantiated.

3. The Uncertain Rule Base

The Uncertain Rule Base contains rules defined by clauses of the simple form:

\[ \text{rule}(N, \text{Conclusion}, \text{List_of_conditions}, \text{Rule_CF}) \]

where:

- \( N \) is any integer, different for different rules and different from any of the model numbers;
- \( \text{Conclusion} \) is a logical string variable. E.g. “Give credit”;
- \( \text{List_of_conditions} \) is a list of logical string variables, e.g. [“Excellent collateral”, “Excellent financial standing”].

\( \text{Rule Cf} \) is the certainty factor associated with the rule \( N \). It is a real string variable being equal either to a real from the interval \([-1,1]\), e.g. “0.65”, or to a name, e.g. “CF_1”, for which the numerical value is determined later either by the user or by a suitable model from the Uncertain Model Base. The \( \text{Rule CF} \) describes the degree of confidence or belief on the Conclusion being true given the certainty factor of the \( \text{List of Conditions} \). More precisely, it describes the influence of the \( \text{List of Conditions CF} \) on the \( \text{Conclusion CF} \). In particular:

a) \( \text{Rule CF}=1 \) indicates that the Conclusion is as certain as the \( \text{List of Conditions} \) is; or the Conclusion is supported entirely by the certainty of the \( \text{List of Conditions} \).

b) \( \text{Rule CF}=-1 \) indicates that the Conclusion is as uncertain as the \( \text{List of Conditions} \) is certain; or the negated Conclusion denoted by \( \neg \text{Conclusion} \) is supported entirely by the certainty of the \( \text{List of Conditions} \).

c) Intermediate values of \( \text{Rule CF} \) indicate intermediate degrees of support.

Obviously, the use of positive and negative certainty factors is rather convenient for modeling conditions supporting the conclusion as well as conditions that don’t.

\( \text{Condition_i CF} \) is the Certainty Factor associated with \( \text{Condition_i} \). It is a real from the interval \([-1,1]\). For askable conditions it is declared by the user; for non-askable conditions – it is calculated by the inference system. \( \text{Condition_i CF}=1 \) indicates that \( \text{Condition_i} \) is absolutely true and \( \text{Condition_i CF}=-1 \) indicates that \( \text{Condition_i} \) is absolutely false.

Intermediate values of \( \text{Condition_i CF} \) indicate intermediate degrees of certainty.

The Certainty Factor Algebra used is a slight modification of the classic Stanford Certainty Factor Algebra, see [2] as quoted in [6]. It may be summarized as follows:

1. Certainty factor for a list of conditions.

Let:

\[ \text{List of conditions} = [\text{"Condition_1"}, \ldots, \text{"Condition_i"}, \ldots, \text{"Condition_n"}] \]

with conditions having certainty factors

\[ \text{"Condition_1 CF"}, \ldots, \text{"Condition_i CF"}, \ldots, \text{"Condition_n CF"} \]

Then:
List_of_Conditions_CF = Minimum(Condition_1_CF, ..., Condition_n_CF)

2) Certainty factor for a single rule
Consider the case when some Conclusion is defined by one rule only:

\[ \text{rule(N, Conclusion, List_of_conditions, Rule_CF)} \]
then:

\[ \text{Conclusion\_CF} = \text{List\_of\_Conditions\_CF} \times \text{Rule\_CF} \]
This establishes the meaning of a single uncertain rule: the Conclusion is true with certainty factor Conclusion\_CF.

If Conclusion\_CF > 0, the rule is said to be uncertainly resolved. If Conclusion\_CF <= 0, the rule is said to be uncertainly unresolved.

3) Multiple rules with the same conclusion
If there are many rules with the same conclusion, they should be designed in one of the following two ways only:

a) rules with the same conclusion have independent lists of conditions, e.g.:

\[ \text{rule(1, "Promotion to Headquarters", ["IQ>100"], "0.4")} \]
\[ \text{rule(2, "Promotion to Headquarters", ["Excellent sales"], "0.7")} \]
The two lists of conditions are clearly independent: there is no junctioin between the managers intelligence quotient and his/her sale performance. Therefore both factors contribute to the certainty of the promotion and both rules must be taken into account.

b) rules with the same conclusion have dependent lists of conditions, e.g.:

\[ \text{rule(1, "Give credit", ["Excellent collateral", "Excellent reputation"], "0.9")} \]
\[ \text{rule(2, "Give credit", ["Good collateral", "Good reputation"], "0.6")} \]
The two lists of conditions are clearly dependent: if the collateral is good, it is not excellent, if the reputation is good, it is not excellent. Obviously, only one of those rules should be applied to establish the certainty factor for giving credit, namely this one, which results in a larger CF for Give credit.

Rules with lists of conditions being partially independent and partially dependent are not permitted.

4) Certainty Factor for two rules with independent list of conditions
Given two rules with the same conclusion and independent lists of conditions:

\[ \text{rule(1, Conclusion, List_of_Conditions_1, Rule_CF_1)} \]
\[ \text{rule(2, Conclusion, List_of_Conditions_2, Rule_CF_2)} \]
with certainty factors for the common conclusion equal respectively

\[ \text{CF_1\_Conclusion and CF_2\_Conclusion} > 0 \]
the overall CF for the conclusion is given by the expression:

\[ \text{CF\_Conclusion} = \text{CF_1\_Conclusion} + \text{CF_2\_Conclusion} - \text{CF_1\_Conclusion} \times \text{CF_2\_Conclusion} \]

However, when CF_1\_Conclusion <= 0 and CF_2\_Conclusion <= 0, the overall CF for the conclusion is equal:

\[ \text{CF\_Conclusion} = \text{CF_1\_Conclusion} + \text{CF_2\_Conclusion} + \text{CF_1\_Conclusion} \times \text{CF_2\_Conclusion} \]
This means that a positive CF for the conclusion from the second rule is increasing the overall certainty factor of the conclusion, whereas a negative CF for the conclusion from the second rule is decreasing it.

Thanks to the presence of the product

\[ \text{CF_1\_Conclusion} \times \text{CF_2\_Conclusion} \]
in the formulae, the overall certainty factor CF\_Conclusion is never larger than 1 or smaller than -1.

The operations described justify yet another important interpretation for CF’s: besides considering them as "measures of truth", they might be looked upon as weighting factors determining the relative influence of the two interpretations of CF may coexist peacefully and profitably in the frame of a single Uncertain Rule Base.

5) Given two rules with the same conclusion and dependent lists of conditions:

\[ \text{rule(N, Conclusion, List_of_conditions_N, Rule_N_CF)} \]
\[ \text{rule(M, Conclusion, List_of_conditions_M, Rule_M_CF)} \]
where rule 1 generates CF_1\_Conclusion, and rule 2 generates CF_2\_Conclusion, the overall CF for the conclusion is equal:

\[ \text{CF\_Conclusion} = \text{Max(CF_1\_Conclusion,CF_2\_Conclusion)} \]
To inform the expert systems that the above two rules with the same conclusion have dependent lists of conditions, the clauses one_rule_only(Conclusion) are being used. By default, all rules with the same conclusion not pinpointed by one_rule_only() clauses, are considered to have independent lists of conditions.

Obviously, the classic Stanford Certainty Factor Algebra has been modified in 4) to make it conform to Aristotelian Logic for the case of CF’s being equal 1 or -1, and in 4) and 5) to give justice to the possible existence of rules with the same condition and either independent or dependent lists of conditions.

6) Using the prefix "n" for negated conclusions and negated conditions, the following relations hold:

If Conclusion\_CF, then nConclusion\_CF = - Conclusion\_CF
and

If Condition\_CF, then nCondition\_CF = Condition\_CF
Because CF values usually result from something that might be considered a rather crude “measurement”, the results of calculations performed upon them should not have an unduly large number of positions right of the decimal points. This is achieved by rounding the results of those operations and presenting them in truncated form with no more than two positions. If, as a result of those operations, the value of CF obtained is equal to 1, it is reduced to 0.99 in order to give to latter non-supporting conditions the chance to decrease the CF value.

The ordering of rules in the Uncertain Rule Base may be any. In particular, rules for the same conclusion need not be adjacent. However, for the sake of clarity and readability, it is recommended to group together rules with the same conclusion. The list of conditions is just a list. The conditions usually have the form “attribute has value”, but no attribute without value need to be defined beforehand. This makes the Uncertain Rule Base flexible, but has to be paid for with defining an Exact Constraint Base and Uncertain Constraint Base, to take account for mutually exclusive “attribute has value” conditions.

Rules of the Uncertain Rule Base may be nested with themselves, i.e. conclusions or negated conclusions of some rules may act as conditions of other rules. Rules of the Uncertain Rule Base may be nested with relational models, i.e. conclusions or negated conclusions of some relational models may act as conditions of rules. The depth of nesting is only limited by the memory available. As a result of nesting, rule conditions may be divided into:

- **Askable Conditions (AC)**, which are neither conclusions of any rules of the Uncertain Rule Base nor conclusions of any relational model of the Model Base. Solely the user determines their CF’s after being asked by the system about their values.
- **Non-Askable Conditions (NAC)**, which are conclusions of some of the rules or conclusions of some relational models. Therefore their CF’s are determined as a result of evaluating those rules or models.

Obviously, in order to start this evaluation, if there is no Uncertain Model Base, the Uncertain Rule Base must contain at least one rule with AC’s only.

### 4. The Uncertain Model Base

The Uncertain Model Base may contain:

1) **basic models** (arithmetical or relational) of the form:

```
    model_b(N, Go_Ahead_Condition, Result/Conclusion, First_Argument, Operation/Relation, Second_Argument, Display_Semaphor, Model_CF),
```

   where:

   - N is the model number, being any integer, different for different models and different from any of the rule numbers.
   - Go_Ahead_Condition is a logical string variables controlling the model application. The model is evaluated only if the Go_Ahead_Condition is true or equal to “no condition”.
   - Result/Conclusion is:
     a) A real string variable (Result) with value computed by Operation performed on First_Argument and Second_Argument of an arithmetical model,
     b) A logical string variable (Conclusion) considered to be true with the certainty factor Model_CF if the model is a relational model and the relation defined by First_Argument, Relation and Second_Argument holds.
   - First_Argument and Second_Argument are real string variables for arithmetical as well as for relational models.
   - Operation/Relation is a string as given below:
     a) for arithmetical models with two arguments: “+”, “-”, “*”, “/”, “mod”, “min”, “max”, “%”;
     c) for relational models: “>”, “<”, “>=”, “<=”, “<>”;
   - Display_Semaphor should be equal 0 if no model evaluation data is to be displayed during reasoning or 1 if otherwise.
   - Model_CF is a real string variable determined as follows:
     a) for arithmetical models it is quite obviously equal to the string “1”;
     b) for relational models it is a string corresponding either to a real from the interval [1,1], e.g. “0.65”, or to a name, e.g. “CF_M12”, for which the numerical value is determined later either by the user or by a suitable model from the Uncertain Model Base. This gives justices to the fact, that some relations (e.g. “>” or “<”) may be fulfilled with various degrees and the user may want to declare the CF with which a relation is fulfilled. Obviously, for relations which do not hold, the Certainty Factor has to be chosen as a equal –1.

2) **extended models** (arithmetical or relational) of the form:

```
    model_e(N, Go_Ahead_Condition, Result/Conclusion, Operation/Relation, List_of_Arguments, Display_Semaphor, Model_CF),
```

   where:

   - Operation/Relation is a string as given below:
     a) for arithmetical models: “+”, “-”, “*”, “/”, “mod”, “min”, “max”, “%”;
     b) for relational models: “<”, “<”, “<”, “<”, “<”, “<”, “<”, “<”;

   ```
   ```
- List_of_Arguments is a list:
  a) of any number of real string variables for an arithmetic extended model;
  b) of three real string variables defining a binary relation:
     [Lower_Bound, Tested_Variable, Upper_Bound] for a relational extended model;

7) linear models of the form:
   model_lin(N, Go_Ahead_Condition, Result, List_of_Coefficients, List_of_Variables, Display_Semaphor, Model_CF),
   where:
   - Result is a real string variable with value computed by evaluating the linear form defined by
     List_of_Coefficients and List_of_Variables;
   - List_of_Coefficients and List_of_Variables are lists with the same number of real string variables;

8) polynomial models of the form:
   model_poly(N, Go_Ahead_Condition, Result, Polynomial_Variable, List_of_Coefficients, List_of_Powers, Display_Semaphor, Model_CF),
   where:
   - Result is a real string variable with value computed by evaluating the polynomial defined by
     List_of_Coefficients and List_of_Powers for some value of the Polynomial_Variable;
   - List_of_Coefficients is a list of real string variables;
   - List_of_Powers is a list of integer string variables in the order corresponding to the order of coefficients in the List_of_Coefficients

Models may be nested inside the Model Base as well as nested with rules of the Rule Base:
   a) results of arithmetical models may serve as arguments for other models;
   b) conclusions of relational models may serve as go ahead conditions for other models;
   c) conclusions of relational models may serve as conditions for rules;
   d) conclusions of rules may serve as go ahead conditions for models.

As a result of nesting, model arguments may be divided into:
Askable Arguments (AA), which are not results of any other models. Solely the user determines their values after being asked by the system to do it.
Non-Askable Arguments (NAA), which are results of some of the models.

Obviously, in order to start the overall evaluation, either the Uncertain Rule Base must contain at least one rule with AC’s only or the Uncertain Model Base must contain at least one model with AA’s only.

5. The Exact Constraint Base

The Exact Constraint Base contains lists of exact mutually exclusive and completely exhaustive AC’s. It is of the form:
   constraint(Constraint_Number, List_of_conditions).
An example is ["Age under 20","Age between 20 and 40","Age above 40"], which contains AC’s for which no uncertainty is conceivable. They are just values for the undefined, implicit attribute “Age_Group”. Obviously, only one condition from such list may have CF=1, while the remaining conditions should have CF=0. The sum of all CF for conditions from such a list should thus be equal either 0 or –1 or –2 etc.

Its main purpose is to block the possibility of asking the user questions about CF’s of AC’s that may be deduced from answers already presented. At the same time it points at rules that should be evaluated and rules that should not. E.g. for the above age constraint, after declaring that CF("Age_under_20")=1 no questions about other age group should be asked. This means that rules with condition “Age_under_20” are evaluated and rules with other age conditions are never.

6. The Uncertain Constraint Base

A set of askable conditions may be mutually exclusive and completely exhaustive in an uncertain way. Consider e.g. the following conditions: "Excellent health", "Good health", "Satisfactory health", "Poor health". They are mutually exclusive in an uncertain way because doctors may differ in their judgement regarding the health of a particular person. To inform the expert system about a situation like this, an Uncertain Constraint Base is created with the uncertain constraint:
   constraint_u(1, ["Excellent health","Good health", "Satisfactory health", "Poor health"]) Generally, the Uncertain Constraint Base contains clauses:
   constraint_u(Constraint_number, List_of_Uncertain_Exclusive_Exhaustive_Conditions)

The user is normally asked to declare certainty factors for all conditions in such clause. Only rules with the condition which got the largest CF are used for reasoning; rules with the remaining conditions are removed from the rule base for the current reasoning.

7. The Advice Base

With each rule there may be associated an advice. The correspondence between rule numbers and advice names is established by the Advice Base, which consists of clauses of the form:
   advice(Rule_Number,Advice_Name)
8. Forward and Backward Chaining

Two different reasoning systems have been developed, supporting Forward Chaining and Backward Chaining. The aim of Forward Chaining is to establish values of CF for all conclusions of the Rule Base and all outputs of the Model Base, given CF's for a selected set of AC's and given values for a selected set of AA's.

Two user interface modes have been implemented for Forward Chaining:

a) The Query Mode, which confronts the user with a series of questions, one at a time, concerning the value of CF for an AC or the numerical value for an AA. The testing starts with the first rule containing only AC's, for which the user is asked to provide CF values. If there is no such rule, it starts with the first model containing only AA's, for which the user is asked to provide argument values. The rule (model) test being accomplished, the reasoning system proceeds to the next rule (model), which either have only AC's (AA's) or which have their NAC's (NAA's) already determined. The CF values for any conclusion are registered in the dynamic database of the system only after all rules for that conclusion have been tested.

b) The AA-AC Menu Mode, confronting the user with a multiple-choice menu for selecting those AA's and AC's, for which the user wishes to declare logical or numerical values of

The aim of Backward Chaining is:

a) to establish the value of CF for a selected hypothesis, which must be the condition of some rule, or
b) to establish the value of CF for a selected hypothesis, which must be the output of a relational or logical model, or
c) to establish the value of some variable, which must be the output of an operational model, given CF's for a selected set of AC's and given values for a selected set of AA's.

The hypotheses have been classified as:

1) Narrow Sense Hypotheses, being all those rule conclusions, which are not conditions of other rules or perhaps models, and those entire model outputs, which are not arguments of other models or conditions of some rules.
2) Broad Sense Hypotheses, being all rule conditions and all model outputs

Both reasoning systems cope successfully with rules and models nested to any depth. No special timesaving mechanisms (see e.g. Nikolopoulos (1997)) have been built into the reasoning system; over 99% of the time to produce results is spend by the system waiting for CF's keyed in by the user and displaying the results obtained.

For Backward Chaining also two interfaces have been implemented: The Narrow Sense Hypotheses Interface and The Broad Sense Hypotheses Interface, each being a single choice menu allowing the user to select the hypothesis to be tested. Selection of the hypothesis starts the testing of all its conditions, the testing of any condition being considered as testing a second order hypothesis, etc. The testing of a hypothesis of any order is completed if its CF values have been determined for all rules, which have as conclusion this hypothesis.

9. Rule – and Constraint Base Verification

The freedom to nest rules brings the danger to build into rule- and constraint bases inconsistencies and redundancies. Inconsistencies lead to performance breakdown and therefore should be removed before starting the reasoning process. Redundancies are less dangerous than inconsistencies: the result usually in wrong CF values and may generate too many conditions for a given conclusion. Their presence has to result in a warning generated just after the base has been loaded. This warning should lead to a detailed diagnostic message pinpointing the culprit rules. As a sequel:

a) in case inconsistencies are discovered, reasoning should be blocked and the user is advised to correct the bases using the diagnostic message detailing the nature of the detected inconsistencies;
b) in case redundancies are discovered reasoning is not blocked, but is discouraged. A diagnostic message detailing the nature of the redundancies is made available to the user.

The following inconsistencies, generated by rule bases solely, are automatically detected and give rise to a diagnostic message:

A. External inconsistencies. They are inconsistencies between the conclusion and conditions of rules.

a) The most elementary is the case of a rule being externally inconsistent like this:

\[ \text{rule}(\_, \text{Conclusion}, [...,	ext{Conclusion},...], \_) \]

or this:

\[ \text{rule}(\_, \text{Conclusion},[....n\text{Conclusion},...], \_) \]

It amounts to the Conclusion (or its negation) being one of the rules condition. This is a canonical case to which all other inconsistencies may be reduced.

b) More difficult to spot is the case of two rules being mutually externally inconsistent:

\[ \text{rule}(1, \text{Conclusion}_1, [...,	ext{Conclusion}_2,...], \_), \] \[ \text{rule}(2, \text{Conclusion}_2, [...,	ext{Conclusion}_1,...], \_) \]

This time the problem is that in order to evaluate \( \text{CF}([\text{Conclusion}_1]) \) from rule 1 we have to know \( \text{CF}([\text{Conclusion}_2]) \) to be determined from rule 2, which can’t be done unless we know \( \text{CF}([\text{Conclusion}_1]) \).

However, the presence of yet another rule for \( \text{Conclusion}_2 \) having no condition \( \text{Conclusion}_1 \):

\[ \text{rule}(3, \text{Conclusion}_2, [....], \_) \]

removes the inconsistency previously discovered.
c) Yet more difficult to diagnose is the case of more rules being indirectly externally inconsistent. This is demonstrated by the example of 3 rules:

\[
\text{rule}(1, \text{Conclusion}_1, [...,\text{Conclusion}_2,...,\_]) \\
\text{rule}(2, \text{Conclusion}_2, [...,\text{Conclusion}_3,...,\_]) \\
\text{rule}(3, \text{Conclusion}_3, [...,\text{Conclusion}_1,...,\_])
\]

In order to evaluate \( \text{CF} (\text{Conclusion}_1) \) from rule 1 we have to know \( \text{CF} (\text{Conclusion}_2) \) to be determined from rule 2, which can’t be done unless we know \( \text{CF} (\text{Conclusion}_3) \), to be determined from rule 3, which can’t be done unless we know \( \text{CF} (\text{Conclusion}_1) \) and the “vicious circle” closes.

**B. Internal inconsistencies.** They are inconsistencies between the conditions of rules.

a) The most elementary is the case of a rule being internally inconsistent like this:

\[
\text{rule} (_{...}, \text{Conclusion}, [...,\text{Condition},...,\text{nCondition},...])
\]

Its essence is that the rule has two mutually exclusive conditions: \( \text{Condition}, \text{nCondition} \). This case is a canonical to which all other internal inconsistencies may be reduced.

b) More difficult to spot is the case of two rules being mutually internally inconsistent:

\[
\text{rule}(1, \text{Conclusion}_1, [...,\text{nCondition}_k,...,\text{Conclusion}_2,...,\_]) \\
\text{rule}(2, \text{Conclusion}_2, [...,\text{Condition}_k,...])
\]

c) Yet more difficult to diagnose is the case of more rules being indirectly internally inconsistent. This is demonstrated by the example of 3 rules:

\[
\text{rule}(1, \text{Conclusion}_1, [...,\text{Conclusion}_2,...,\_]) \\
\text{rule}(2, \text{Conclusion}_2, [...,\text{Conclusion}_3,...,\_]) \\
\text{rule}(3, \text{Conclusion}_3, [...,\text{Conclusion}_1,...,\_])
\]

Because of rule 2, the conditions of rule 1 effectively include \( \text{Condition}_k \) as well as \( \text{nCondition}_k \), the result being an inconsistent set of conditions.

Inconsistencies may also be generated as a result of interaction between rules and constraints. Consider the following example:

\[
\text{rule}(1, \text{Conclusion}_1, [...,\text{Condition}_1,...,\text{Conclusion}_2,...,\_]) \\
\text{rule}(2, \text{Conclusion}_2, [...,\text{Conclusion}_2,...,\_]) \\
\text{constraint}(1,[...,\text{Condition}_1,...,\text{Condition}_2,...])
\]

Due to constraint 1, rule 1 will never be tested: either because \( \text{Condition}_1=\_1 \) or because \( \text{Condition}_2=\_1 \)

Redundancies occur if more rules have the same conclusion. Their detection is a subtler problem, difficult to solve without inquiring into the designer intentions. Consider e.g. the following rules with the same conclusion:

\[
\text{rule}(1, \text{Conclusion}, [...,\text{Condition}_1,...,\text{Condition}_2,...]) \\
\text{rule}(2, \text{Conclusion}, [...,\text{Condition}_1,...,\text{Condition}_2,...])
\]

The intention might be to stress the fact, that the additional \( \text{Condition}_3 \) increases the certainty of \( \text{Conclusion} \). This however has been done in a way that makes it difficult to estimate this increase. It would be better to do it perhaps this way:

\[
\text{rule}(1, \text{Conclusion}, [...,\text{Condition}_1,...,\text{Condition}_2,...]) \\
\text{rule}(2, \text{Conclusion}, [...,\text{Condition}_3,...])
\]

However, for the case of the following two rules:

\[
\text{rule}(1, \text{Conclusion}, [...,\text{Condition}_1,...,\text{Condition}_2,...]) \\
\text{rule}(2, \text{Conclusion}, [...,\text{Condition}_1,...,\text{Condition}_3,...])
\]

rule 2 might be safely considered to subsumed by rule 1 and therefore removed from the Rule Base.

A basic redundancy occurs for rules with the same conditions and the same conclusion. It is however sometimes created by the interaction of many presumbable good-looking rules, like e.g.:

\[
\text{rule}(1, \text{Conclusion}, [...,\text{Condition}_1,...,\text{Condition}_2,...]) \\
\text{rule}(2, \text{Conclusion}, [...,\text{Condition}_1,...,\text{Condition}_2,...]) \\
\text{rule}(3, \text{Conclusion}, [...,\text{Condition}_1,...,\text{Condition}_2,...]) \\
\text{rule}(4, \text{Conclusion}, [...,\text{Condition}_1,...,\text{Condition}_2,...])
\]

Rule 4 may be saying exactly the same as rules 1, 2 and 3, perhaps with a different certainty factor. The designer should be asked whether this is really intended.

Another basic redundancy occurs for rules with unnecessary conditions, like e.g.:

\[
\text{rule}(1, \text{Conclusion}, [...,\text{Condition}_1,...,\text{Condition}_2,...]) \\
\text{rule}(2, \text{Conclusion}, [...,\text{Condition}_1,...,\text{Condition}_2,...])
\]

This time, dependent upon the condition \( \text{CF} \) values, both rule say exactly the same thing (and this is a bad thing because the combined \( \text{CF} \) for the \( \text{Conclusion} \) unnecessarily increases) or almost cancel each other (and this is bad as well). The best thing to do seems to remove one of those rules.

Redundancies may be created also as the result of interaction of rules and constraints. E.g. the following two rules and single constraint:

\[
\text{rule}(1, \text{Conclusion}, [...,\text{Condition}_1,...,\text{Condition}_2,...]) \\
\text{rule}(2, \text{Conclusion}, [...,\text{Condition}_1,...,\text{Condition}_2,...]) \\
\text{constraint}(1,[...,\text{Condition}_3,...])
\]

This time, dependent upon the condition \( \text{CF} \) values, both rule say exactly the same thing (and this is a bad thing because the combined \( \text{CF} \) for the \( \text{Conclusion} \) unnecessarily increases) or almost cancel each other (and this is bad as well). The best thing to do seems to remove one of those rules.
Developing the system for automatic detection of inconsistencies and redundancies turned out to be quite an ordeal, surpassing in difficulty the job of building the proper reasoning system.

10. Example: Credit Evaluation Knowledge Base

Credit evaluation is a good testing ground for rule- and model based expert systems: rules, models and facts are needed to establish the trustworthiness of bank clients. The reason for using expert systems for such a task is obvious: the evaluation is normally done on a massive scale and the rules, with which it is done, are changing with changing market conditions; therefore, to put the domain knowledge into a some standard procedural program, would result in rather costly updating and maintenance work, to say nothing about the poor readability of the knowledge contained in a procedural program.

The Credit Evaluation Knowledge Base considered allows to generate CF’s for one of the following two decisions: 1) Give credit, 2) Consult with boss.

Three basic credit factors are taken into account to reach the decision: A) Collateral, B) Financial standing, C) Reputation

Ad A) Collateral is considered by the Bank as basic factor influencing the credit decision. The Bank recognises the following categories:

A1. Collateral is excellent, if:
   a) 100 <= I-class collateral (%)  
      where:
      I-class collateral (%) = Local currency deposits + USD deposits + Bank Guarantees 
      referred to the credit required and expressed in %,  
      or
   b) b1) 70 <= I-class collateral (%) < 100 and  
       b2) 30 <= II-class collateral (%)  
      where:
      II-class collateral (%) = Stocks + Bonds  
      referred to the credit required and expressed in %,

A2. Collateral is good, if:
   1. 60 <= I-class collateral (%) and 
   2. 10 <= II-class collateral (%) and 
   3. 30 <= III-class collateral (%) and 
   4. Collateral is not excellent 
   where:
   III-class collateral (%) =  
   Real Estate Mortgage + Property Rights 
   referred to the credit required and expressed in %,

A3. Collateral is bad, if it is not excellent and not good.

Ad B) The financial standing is evaluated using four parameters:

B1. Short-term debts/Net sale (in %), with weighting factor -2  
B2. Last year increase of profit (in %), with weighting factor 1  
B3. Net profit/Total assets (in %), with weighting factor 5  
B4. Net profit/Net sales (in %), with weighting factor 5

The weighted sum of these parameters gives the Financial Indicator. The financial standing is considered:
   1) Excellent, if Financial Indicator > 300  
   2) Good, if 100 < Financial Indicator <= 300  
   3) Passable, if -50 < Financial Indicator <= 100  
   4) Bad, if Financial Indicator <= -50

Ad C) Reputation is evaluated using the following categories:

   Excellent reputation
   Good reputation
   Bad reputation

Credits are given provided the collateral, financial standing and reputation are at least good. 

No credits are given to clients with bad financial standing, or with bad collateral, or with passable financial standing and bad reputation.

In the remaining cases, if the clients reputation is bad or the financial standing is passable, the boss of the loan officer needs to be consulted.

The need to use certainty factor arose from the necessity to have a more precise measure of the clients status. E. g. the collateral pf two clients may satisfy all inequalities needed to classify them as “Good collateral”, for the first client with an ample margin, however for the second client by the skin of our teeth. This difference is made evident by differentiating the certainty factors for those inequalities for the first and second client.

Altogether, the entire ‘decision space” has the following "number of points":
   3 (collateral) * 4 (Financial Standing) * 3 (Reputation) = 36

However, by proper aggregation of these points, we make do with only 20 rules connected strictly with credit decisions.

The Uncertain Rule Base contains following rules:

rule(1,"Give credit","[Excellent collateral","Excellent financial standing","Excellent reputation"],"0.9")  
rule(2,"Give credit","[Excellent collateral","Excellent financial standing","Good reputation"],"0.8")  
rule(3,"Give credit","[Excellent collateral","Good financial standing","Excellent reputation"],"0.8")  
rule(4,"Give credit","[Excellent collateral","Good financial standing","Good reputation"],"0.7")  
rule(5,"Give credit","[Good collateral","Excellent reputation"],"0.7")  
rule(6,"Give credit","[Good collateral","Good financial standing","Excellent reputation"],"0.6")  
rule(7,"Give credit","[Good collateral","Good financial standing","Good reputation"],"0.5")  
rule(8,"Give credit","[Good collateral","Passable financial standing","Excellent reputation"],"0.4")  
rule(9,"Give credit","[Good collateral","Passable financial standing","Good reputation"],"0.3")  
rule(10,"Give credit","[Good collateral","Passable financial standing","Passable reputation"],"0.2")  
rule(11,"Give credit","[Good collateral","Passable financial standing","Bad reputation"],"0.1")  
rule(12,"Give credit","[Good collateral","Bad financial standing","Excellent reputation"],"0.05")  
rule(13,"Give credit","[Good collateral","Bad financial standing","Good reputation"],"0.05")  
rule(14,"Give credit","[Good collateral","Bad financial standing","Passable reputation"],"0.05")  
rule(15,"Give credit","[Good collateral","Bad financial standing","Bad reputation"],"0.05")  
rule(16,"Give credit","[Bad collateral","Excellent financial standing","Excellent reputation"],"0.05")  
rule(17,"Give credit","[Bad collateral","Excellent financial standing","Good reputation"],"0.05")  
rule(18,"Give credit","[Bad collateral","Excellent financial standing","Passable reputation"],"0.05")  
rule(19,"Give credit","[Bad collateral","Excellent financial standing","Bad reputation"],"0.05")  
rule(20,"Give credit","[Bad collateral","Good financial standing","Excellent reputation"],"0.05")  
rule(21,"Give credit","[Bad collateral","Good financial standing","Good reputation"],"0.05")  
rule(22,"Give credit","[Bad collateral","Good financial standing","Passable reputation"],"0.05")  
rule(23,"Give credit","[Bad collateral","Good financial standing","Bad reputation"],"0.05")  
rule(24,"Give credit","[Bad collateral","Passable financial standing","Excellent reputation"],"0.05")  
rule(25,"Give credit","[Bad collateral","Passable financial standing","Good reputation"],"0.05")  
rule(26,"Give credit","[Bad collateral","Passable financial standing","Passable reputation"],"0.05")  
rule(27,"Give credit","[Bad collateral","Passable financial standing","Bad reputation"],"0.05")  
rule(28,"Give credit","[Bad collateral","Bad financial standing","Excellent reputation"],"0.05")  
rule(29,"Give credit","[Bad collateral","Bad financial standing","Good reputation"],"0.05")  
rule(30,"Give credit","[Bad collateral","Bad financial standing","Passable reputation"],"0.05")  
rule(31,"Give credit","[Bad collateral","Bad financial standing","Bad reputation"],"0.05")  
rule(32,"Give credit","[Bad collateral","Bad collateral","Excellent reputation"],"0.05")  
rule(33,"Give credit","[Bad collateral","Bad collateral","Good reputation"],"0.05")  
rule(34,"Give credit","[Bad collateral","Bad collateral","Passable reputation"],"0.05")  
rule(35,"Give credit","[Bad collateral","Bad collateral","Bad reputation"],"0.05")  
rule(36,"Give credit","[Bad collateral","Bad collateral","Excellent reputation"],"0.05")  
rule(37,"Give credit","[Bad collateral","Bad collateral","Good reputation"],"0.05")  
rule(38,"Give credit","[Bad collateral","Bad collateral","Passable reputation"],"0.05")  
rule(39,"Give credit","[Bad collateral","Bad collateral","Bad reputation"],"0.05")  
rule(40,"Give credit","[Bad collateral","Bad collateral","Excellent reputation"],"0.05")  
rule(41,"Give credit","[Bad collateral","Bad collateral","Good reputation"],"0.05")  
rule(42,"Give credit","[Bad collateral","Bad collateral","Passable reputation"],"0.05")  
rule(43,"Give credit","[Bad collateral","Bad collateral","Bad reputation"],"0.05")
The Uncertain Model Base contains following models:

- `model_b(101,"no condition","Excellent collateral a")`,
  - "Excellent collateral a")", ","="",100",1,"CF_EX_C"

- `model_e(102,"no condition","Excellent collateral b1)",
  - "<="",70",1,"CF_P1_C"

- `model_b(103,"Partial excellent collateral b1)",
  - "Excellent collateral a")", ","="",100",1,"CF_P1X_C"

- `model_b(104,"Partial excellent collateral b1)",
  - "Partial good collateral 1")","="",60",1,"CF_P1G_C"

- `model_b(105,"Partial good collateral 1)",
  - "Partial good collateral 2")","="",10",1,"CF_P2G_C"

- `model_b(106,"Partial good collateral 2)",
  - "Partial good collateral 3")","="",30",1,"CF_P3G_C"

- `model_e(107,"no condition","I-class collateral ", ",","+",
  - "Local currency deposits":"USD deposits",
  - "Bank guarantees":1,"1.0")

- `model_b(108,"no condition","I-class collateral ",
  - "I-class collateral ",","Credit required":1,"1.0")

- `model_b(109,"no condition","II-class collateral",
  - "Stock","+","Bonds":1,"1.0")

- `model_b(110,"no condition","II-class collateral ",
  - "II-class collateral ",","Credit required":1,"1.0")

- `model_b(111,"no condition","III-class collateral",
  - "Real estate mortgage","+","Property rights":1,"1.0")

- `model_b(112,"no condition","III-class collateral ",
  - "III-class collateral ",","Credit required":1,"1.0")

- `model_b(113,"no condition","Financial indicator ",
  - ">=",2",1","5","5")

- `model_e(114,"no condition","Financial indicator ",
  - "<="",50",1,"CF_BFS")

- `model_e(115,"no condition","Passable financial standing ",
  - "<="",100",1,"CF_PFS")

- `model_e(116,"no condition","Good financial standing ",
  - "<="",-50",1,"CF_GFS")

Numerical values for Certainty Factors of relational models are determined by the user in the process of reasoning. E. g. consider model 116: if the Financial Indicator is near to the upper bound, the value of CF_GFS may be chosen as 0.8; if it is near to the lower bound, the value of CF_GFS may be chosen as 0.2; Only if the relation from model 116 is not fulfilled, CF_GFS must be chosen equal –1.

The Uncertain Constraint Base contains one constraint only:

- `constraint(1,"Excellent reputation","Good reputation","Bad reputation")`

The user must for at least one of those conditions declare a CF>0 value. From all those rules containing the conditions from the Uncertain Constraint Base, only those rules are taken into account, which contain the exclusive condition with the greatest CF value.
11. Conclusions

The expert system shells described served – at various stages of their development – as vehicle for teaching students the art of modelling and solving decision problems by means of knowledge bases to be used for inference with the shells. The development confirmed the initial assumption of considering Prolog as a most suitable tool for writing programs with explicit domain knowledge representation.

Acknowledgement

It is kindly acknowledged that this development has been supported from March 2001 by the Foundation for Polish Science under the Subsidy for Scientists 10/20001

BIBLIOGRAPHY

[16] PDC 331A Prolog Users Guide. Prolog Development Center, Copenhagen, 1993, see http://www.pdc.dk/vip