

Extending Jess to Handle Uncertainty

David Corsar, Derek Sleeman, and Anne McKenzie
Department of Computing Science
The University of Aberdeen
Aberdeen, UK
{dcorsar, d.sleeman}@abdn.ac.uk

Abstract

Computer scientists are often faced with the challenge of having to model the world and its associated uncertainties. One area in particular where modelling uncertainty is important are Expert Systems (also referred to as Knowledge Based Systems and Intelligent Systems), where procedural / classification knowledge is often captured as facts and rules. One of the earliest Expert Systems to incorporate uncertainty was MYCIN. The developers realized that uncertainty had to be associated with both the properties of the objects they were modelling and with the knowledge (the rules themselves). A popular engine for building Knowledge Based Systems currently is Jess, which has been extended to handle uncertain knowledge by using fuzzy logic. However, systems written using this extension are generally composed of two interrelated components – namely a Java program and a Jess knowledge base. Further, this technique has several other disadvantages which are also discussed. We have developed a system, Uncertainty Jess, which provides Jess with the same powerful, yet easy to use, uncertainty handling as MYCIN. Uncertainty Jess allows the user to assign certainty factors / scores to both the properties of their data and to the rules, which it then makes use of to determine the certainty of rule conclusions for single and multiple identical conclusions.

1. Introduction: Handling Uncertainty

Humans are good at handling uncertainty in both their everyday and professional lives, and so we perhaps underplay its pervasiveness and importance. There are many ways in which scientists and more recently computer scientists have attempted to model the world and its associated uncertainties. Many approaches to modelling devised by computer science, make the strong distinction between objects which have properties and values, and knowledge which says whether and how those objects should be classified / manipulated when certain conditions (on the objects themselves) are satisfied. One such approach is Expert Systems (also referred to as both Knowledge Based Systems (KBSs) and Intelligent Systems) which were introduced in the 1970s [4]. Here the procedural / classification knowledge is often captured as a rule. To effectively model domains, in this formalism, the developers realized that uncertainty had to be associated with both

the properties of the object and with the knowledge (the rules themselves). In the next section (section 2) we discuss a concrete example from the domain of medical decision making, but in general such a formalism needs to address a variety of issues including:

- The need to represent uncertainty in both knowledge and data.
- The details of how such uncertainties are represented
- How the different pieces of evidence are combined.
- How the certainty level of an outcome affects decisions that the Knowledge Based System makes.

We recently developed a system to help medical technicians classify organisms present in blood samples, [5]. The decision was made to use the Jess system [3] as it was readily available here. However, it was soon clear that it was going to be essential in this domain to handle the inherent uncertainty in the *data* being provided by the technicians about various aspects of blood samples (for example the *shape* of the organism). Additionally, the pathologist was also clear that the classificatory rules used in this domain were always associated with uncertainties. That is, even if all the data values could be precisely determined, a range of diseases would virtually always be returned (each associated with a likelihood.) Thus if we were to use the Jess inference engine for this task it was essential that we were able to handle uncertainties at both these levels. This paper describes these investigations.

The structure of the rest of this paper is: section 2 gives an overview of how an early Expert System, MYCIN, handled uncertainty; section 3 give a brief overview of the Jess rule language; section 4 discusses extensions to Jess to enable it to handle uncertainty in a MYCIN-like way; section 5 presents alternative approaches for handling uncertainties in Jess; section 6 compares the several approaches for handling uncertainties; and section 7 provides conclusions and suggestions for future work.

2. Overview of how MYCIN Handled Uncertainty

As noted in the introduction, in many scientific domains there is uncertainty associated with the data which a user is able to provide and also in the domain knowledge. Below, we give a typical MYCIN rule [1] which classifies an organism (in this instance as enterobacteriaceae):

```
IF      the stain of the organism is gram negative
AND    the morphology of the organism is rod
AND    the aerobicity of the organism is aerobic
THEN   the class of the organism is enterobacteriaceae with confidence 0.9
```

This rule has 3 conditions, and if they are all satisfied then the rule is said to be satisfied when it reports a conclusion (the class of the organism is enterobacteriaceae) with a confidence level of .9, on a confidence scale of -1 (totally negative) to +1 (totally positive). MYCIN uses the term, Certainty Factor (CF), to represent confidence levels. These are discussed in more detail in Figure 2.1.

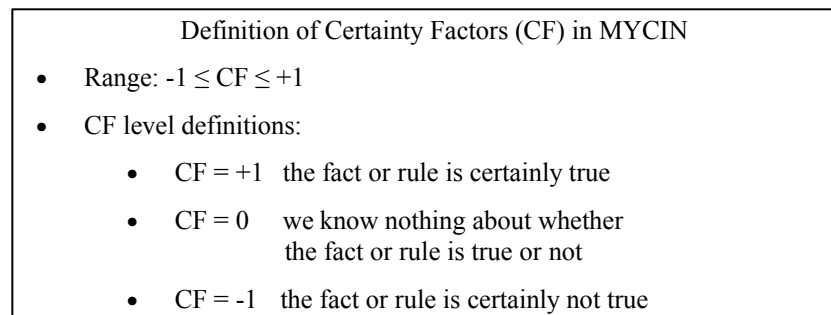


Figure 2.1: A Summary of Certainty Factors in MYCIN

2.1 Uncertainty in MYCIN: Data and Rules

When the user inputs values which correspond to the three conditions in the rule shown above he is asked to specify his confidence in the data which he is reporting. Suppose the user has reached the following conclusions about the data (the organism he is analysing under a microscope):

- The stain of the organism is definitely gram negative (CF = 1.0)
- The morphology is rod, with confidence (CF = 0.8) **OR**
- The morphology is coccus, with confidence (CF = 0.2)
- The aerobicity is aerobic, with confidence (CF = 0.6) **OR**
- The aerobicity is anaerobic, with confidence (CF = 0.3) **OR**
- The aerobicity is both, with confidence (CF = 0.1)

Then when asked his confidence in the three conditions in the rule he would answer 1.0, 0.8 & 0.6 respectively. Further, MYCIN has a procedure for calculating the CF for a conclusion of a rule on the basis of the strength of belief in the rule itself and in the various data inputs:

$CF_{\text{conclusion}} = CF_{\text{rule}} * CF_{\text{data}}$, and

$CF_{\text{data}} = \min(CF_{d1}, CF_{d2}, \dots, CF_{dn})$ where the CF_{di} are the CFs for the several data inputs (Formula [I])

In this instance:

$$CF_{\text{rule}} = 0.9$$

$$CF_{d1} = 1.0, CF_{d2} = 0.8, \text{ and } CF_{d3} = 0.6.$$

So

$$CF_{\text{conclusion}} = .9 * \min(1.0, .8, .6)$$

$$CF_{\text{conclusion}} = .9 * .6 = .54$$

2.2 Uncertainty in MYCIN: Multiple Conclusions

Not infrequently in rule bases like MYCIN's several rules reach the same conclusion. In fact one could argue that this is the normal situation in KBSs, where a variety of data sources and rules are required to be considered before one is able to reach a conclusion with any level of certainty, as generally KBSs are used in situations where deterministic algorithms do not exist. For these reasons MYCIN needed a mechanism to calculate the cumulative CF for a conclusion reached by several rules. So suppose one had a KB with the following rules:

Rule 1: IF A THEN B

Rule 2: IF C AND D AND E THEN B

The algorithm specifies that one should:

1. Calculate the CF for each rule (i.e. each conclusion, in this case B)
2. Combine the various CFs for that conclusion, using the procedure given below:
 - Suppose a rule reports a conclusion with certainty CF_p and another rule reaches the same conclusion with certainty CF_n . How the combined CF (CF_{comb}) is calculated depends on the signs of CF_p and CF_n
 - IF $CF_p > 0$ & $CF_n > 0$ THEN $CF_{\text{comb}} = CF_p + CF_n - CF_p * CF_n$
 - IF $CF_p < 0$ & $CF_n < 0$ THEN $CF_{\text{comb}} = CF_p + CF_n + CF_p * CF_n$
 - ELSE $CF_{\text{comb}} = (CF_p + CF_n) / (1 - \min(\text{abs}(CF_p), \text{abs}(CF_n)))$

(Formula [II])

Figure 2.2 shows two rules which both have conclusions B. The two rules in this figure are in fact:

Rule1: IF A THEN CONCLUSION B with confidence 1.0

Rule2: IF C AND D AND E THEN CONCLUSION B with confidence 0.9

The above procedure indicates how to determine the conclusion and confidence of both of the rules (using Formula [I]) and then how to combine these confidences (using Formula [II]). So the result of this calculation is .862; the calculations are in fact summarized in Figure 2.2.

Further, if a third rule also concluded B with a CF of .6 then the overall CF would then become:

$$.862 + .6 - .862 * .6 = .9448$$

and so on for each additional rule which concludes B. Note, too that if a particular CF (say CF_2) = 1 then the CF_{comb} will equal 1 as $(x + 1 - x * 1 = 1)$ (i.e. positive certainty). Analogously, if the $CF = -1$ then CF_{comb} can be shown to be -1 (i.e., negative certainty).

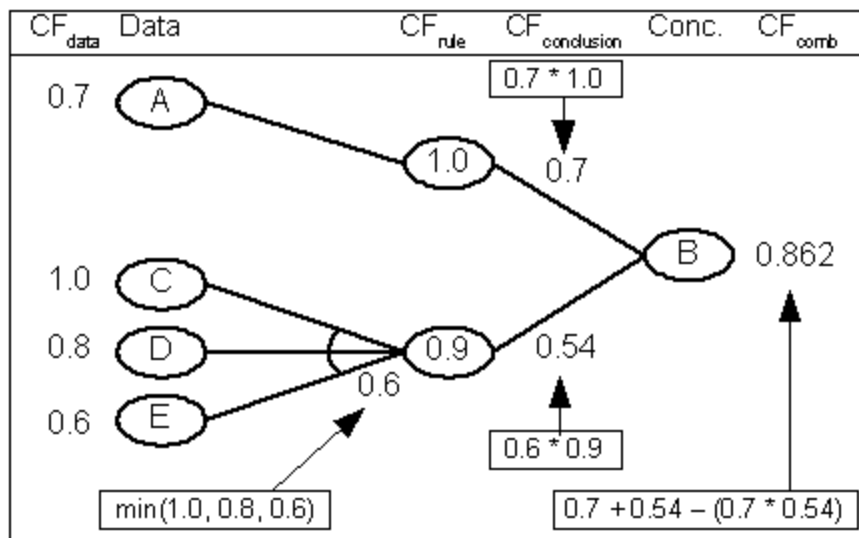


Figure 2.2: Illustration of how the CFs for data A (0.7), C (1.0), D (0.8), and E (0.6) are combined with the CF for rules Rule 1 (1.0) and Rule 2 (0.9), to give CFs for the conclusion B (0.7 for Rule 1, 0.54 for Rule 2) and how these are combined to give an overall CF for conclusion B (0.862).

So the overall algorithm for calculating the CF for a conclusion is:

- Follow up all conclusions regardless of their Certainty Factors
- Note that we cannot simply take the first chain that provides support, as the next could give more (or less) confidence in the result.
- The stopping condition is when the complete space has been searched OR we find a confidence level of 1.0 OR -1.0.

3. Brief Introduction to Jess

Jess:

- Is a forward chaining rule engine implemented in Java; it is a partial reimplementation / redevelopment of the CLIPS [2] Expert System shell.
- Stores facts in working memory (WM); decisions are made by running rules against the facts currently in WM.
- Allows one to define deftemplates, which specify the structure of facts, and deffacts which assert a series of facts (instances of the deftemplates) to WM.
- Allows users to define rules.
- Allows users to define functions directly in Jess; and user-functions, which are Java classes written externally to Jess, but that can be called using the standard Jess syntax.
- Provides the reasoning engine for JessTab, a plug-in for the Protégé ontology editor [7], which allows Jess rules to be executed against a populated Protégé ontology.

The Jess system is implemented in Java which makes it relatively straight forward to create Java programs which incorporate the Jess inference mechanism. For example, the Haematology Diagnostic Assistant developed in Aberdeen, has a Java front-end which acquires information from the user (e.g., level of certainty of data), passes it to the Jess system, which performs reasoning on it; the results are then returned to the Java front-end which displays the results to the user, [5].

For more details of the Jess language and interpreter, see [3].

4. Extending Jess to handle Uncertainty in a MYCIN-like way

The example discussed in section 2 is one of a number of rules that would be included in a system such as MYCIN. In this section we discuss how one would implement such a rule system in Jess; initially with a simple rule which excludes certainty factors, then one which includes them.

To implement the organism classification described previously, one is required to provide a deftemplate which describes the data that is being observed, and the organisms' classifications; a series of facts which provide the details of the actual observations; and some rules for generating classifications. Sample versions of these are provided in Figure 4.1.

This provides a simple (deterministic) implementation for classifying organisms of the enterobacteriaceae class, if they are of the gram negative strain, have a rod morphology and aerobic aerobicity. This is however, only part of the story: the example provided previously states that this classification only has a certainty of

0.9: however the implementation shown in Figure 4.1 does not take this into account.

To include uncertainty in the style of MYCIN into the system shown in Figure 4.1, we are required to make four changes:

1. The deftemplates for observations and classifications need to include a slot in which the certainty factor can be provided.
2. The rule then needs to be modified to generate a certainty factor for the classification.
3. Another rule is required which detects multiple identical conclusions and calls a function to calculate the certainty factor of multiple identical conclusions.
4. A function would be useful for calculating the combined certainty factor of multiple conclusions.

```
(deftemplate observable "The value of a property based on the
clinicians observations"
  (slot name) ; the name of the property being describes
  (slot value) ; the state, or value attached to the property
)
(deftemplate classification "The class to which the observed
organism has been classified"
  (slot class-name)
)
(deffacts some-observations
  (observable (name strain) (value GramNegative))
  (observable (name morphology) (value rod))
  (observable (name aerobicity) (aerobic))
)
(defrule enterobacteriaceae-classification
  (observable (name strain) (value GramNegative))
  (observable (name morphology) (value rod))
  (observable (name aerobicity) (aerobic))
=>
  (assert (classification (class-name enterobacteriaceae) ) )
)
```

Figure 4.1: Example deftemplates and rule for enterobacteriaceae classification.

Changes 1 and 2 are relatively trivial steps, which the developer of the knowledge base would be expected to do. Change 3 is partially generic; so we provide a rule which the developer has to tailor to their own deftemplate used for classifications / conclusions; change 4 is generic enough for us to provide an implementation which developers can simply include in their system. The user / developer is then required to provide certainty factors for both the data (observations in Figure 4.1) and the rules. When defining the uncertainty factors for the data, a user interface

can prompt the user to provide the necessary certainty factors, or these can be included directly in the deffacts. It is slightly different with the rules, however, in this application, the certainty factor must be included when each rule is created. Figure 4.2 shows the deftemplates, facts and rules initially introduced in Figure 4.1, extended with uncertainty data. Figure 4.3 shows a customisable rule for detecting multiple identical conclusions (classifications in this application); Figure 4.4 shows the Jess version of the generic function for calculating the certainty factor of multiple conclusions (deffunction calculate-combined-uncertainty). This system for MYCIN-like uncertainty reasoning in Jess is available as Uncertainty Jess.

```
(deftemplate observable "The value of a property based on the
clinicians observations"
  (slot name) ; the name of the property being described
  (slot value) ; the state, or value attached to the property
  (slot certainty-factor) ; the certainty of this observation
)

(deftemplate classification "The class to which the observed
organism has been classified"
  (slot class-name)
  (slot certainty-factor) ; the CF of this classification
)

(deftemplate combined-classification "Stores the result of
combining several classifications for the same class-name"
  (slot class-name) (slot certainty-factor)
)

(deffacts some-observations
  (observable (name strain) (value GramNegative) (certainty-
factor 1.0))
  (observable (name morphology) (value rod) (certainty-factor
0.8))
  (observable (name aerobicity) (value aerobic) (certainty-
factor 0.6))
)

(defrule enterobacteriaceae-classification "a rule for
classification of enterobacteriaceae, including certainty
factors"
  ; variables in Jess start with ?var-name
  ; store the certainty-factors of the observations in variabes
?scf, ?mcf and ?acf
  (observable (name strain) (value GramNegative) (certainty-
factor ?scf))
  (observable (name morphology) (value rod) (certainty-factor ?
mcf))
  (observable (name aerobicity) (value aerobic) (certainty-
factor ?acf))
  =>
  ; calculate CFConclusion as 0.9 (the CF for this rule) *
minimum CF of the data (stored in variables ?scf, ?mcf and ?
acf), store it in the ?cfconc variable
  (bind ?cfconc (* 0.9 (min ?scf ?mcf ?acf) ) )
  (assert (classification (class-name enterobacteriaceae)
(certainty-factor ?cfconc)) )
)
```

Figure 4.2: The example deftemplates and rules from Figure 4.1 extended with CF information.

```

(defrule detect-and-calculate-multiple-conclusions "Combines all
the certainty factors for classifications with the same class-
name"
  ; declaring the salience to be -10 (the default for rules is
0) means this rule runs after all other rules have completed
  (declare (salience -10))
  ; find a classification with a certain name, stored in ?name
variable
  (classification (class-name ?name))
  ; create a new variable ?c, which will be a list of all the
certainty factors associated with classifications with the
class-name ?name
  ; the Jess accumulate function, first creates a new (empty
list), stored in ?cfs
  ?c <- (accumulate (bind ?cfs (create$))
  ; this line gets executed after the 'body' of the function and
adds ?cf to the list of certainty factors (?cfs)
    (bind ?cfs (create$ ?cfs ?cf))
    ?cfs ; this value gets returned
  ; the body of the function - find a classification with
class-name ?name and store its certainty-factor in ?cf
    (classification (class-name ?name) (certainty-factor ?cf)))
  =>
  ; ?c is now a list of certainty factors
  ; assert the combined classification for all the certainty
factors in ?c, by using the function calculate-combined-
certainty to calculate their combined certainty factor
  (assert (combined-classification (class-name ?name) (certainty-
factor (calculate-combined-uncertainty ?c) ) ) )
)

```

Figure 4.3: A customisable rule which detects if multiple classifications to the same class have been concluded.

```

; This function calculates the combined uncertainty of the
values passed in the multislot CFConcs variable (a multislot
variable is a list variable in Jess)
(deffunction calculate-combined-uncertainty ($?CFConcs)
  // convert the first certainty factor in $?CFConcs to a float,
and store in ?cf
  (bind ?cf (float (nth$ 1 ?CFConcs)))
  // for every remaining certainty factor in $?CFConcs
  (foreach ?cfc (rest$ ?CFConcs)
    // convert the current certainty factor to a float
    (bind ?cfconc (float ?cfc))
    // combine the current ?cf with ?cfconc according to Formula
[II]
    (if (and (> ?cf 0.0) (> ?cfconc 0.0)) then
      (bind ?cf (- (+ ?cf ?cfconc) (* ?cf ?cfconc)))
    else (if (and (< ?cf 0.0) (< ?cfconc 0.0)) then
      (bind ?cf (+ (+ ?cf ?cfconc) (* ?cf ?cfconc)))
    else (bind ?cf (/ (+ ?cf ?cfconc) (- 1.0 (min (abs ?cf)
(abs ?cfconc))))))
  )
  )
  ; return the final certainty factor value - ?cf
  ?cf
)

```

Figure 4.4: A generic function, calculate-combined-uncertainty, which calculates the combined uncertainty of several CFs.

5. Alternative Approaches to Handling Uncertainties in Jess

We have presented one approach to handling uncertainties in Jess, based on the use of certainty factors as implemented in the MYCIN system. An alternative method for dealing with uncertain knowledge in Jess is to use the FuzzyJess extension, [6]. Part of the FuzzyJ toolkit, FuzzyJess is built upon the concept of fuzzy logic and fuzzy set theory whereby knowledge is described in terms of its degree of membership of a fuzzy set. Membership is defined as a value between zero and one, where a value of zero indicates something is definitely not a member of that fuzzy set, and a value of one means it definitely is a member of that set¹. FuzzyJess deals with three main concepts: fuzzy variables, fuzzy sets and fuzzy values. A fuzzy variable defines the basic components used to describe a fuzzy concept, providing the name of the variable, the units of the variable (if required), and the variables' universe of discourse. A fuzzy set defines a mapping of a real number onto a membership value in the range zero to one. A fuzzy value is an association of a fuzzy variable, a fuzzy set and a linguistic expression which describes the fuzzy variable. If a fuzzy value falls within the boundaries of the fuzzy set it is said to be a member of that set. Before a fuzzy value can be output from the system it has to be transformed to an actual value through the process of defuzzification.

The problem with using FuzzyJess when dealing with uncertainty is that it does not naturally support certainty factors. This makes implementation more complex than the MYCIN style presented in section 4. The user is required to define various fuzzy variables that will be relevant to their application, along with relevant fuzzy values for describing the variables' universe of discourse. An example for defining certainty is provided in Figure 5.1.

Figure 5.1 also includes an example rule for the classification of enterobacteriaceae. As can be seen, one of the other problems is that FuzzyJess requires the rules to reference Java code to create the various fuzzy objects, which requires the author to be familiar with the FuzzyJess API, which is far larger and more complex than our Uncertainty Jess implementation, and so will require longer for the author to become familiar with. One of the other major problems is the number of rules that need to be written: in our approach, as long as there are the three observables for strain, morphology and aerobicity, the classification rule always fires, regardless of the certainty factors. The fuzzy-enterobacteriaceae-classification rule (Figure 5.1) on the other hand, only fires when the various certainty factors match (according to the fuzzy-match method) a certain fuzzy value. To ensure the classification is always generated, multiple versions of the rule will be required, dealing with the various different FuzzyValues each certainty-factor could have. For v observables, being matched against a single FuzzyVariable with v FuzzyValues, a minimum of v^0 rules will be required. Another problem with FuzzyJess is that there does not seem to be any way of combining the strength of belief of multiple conclusions (which is possible in the MYCIN approach). That is, if several rules are fired and produce the same conclusions (with different or the same fuzzy output values), there is no built-in

¹ Note this system uses a different scale from that used by the MYCIN CFs.

feature for combining them; one approach would be to provide a series of rules, one for each combinations of FuzzyValues, however, this approach obviously does not scale.

```
; create a new FuzzyVariable for Certainty, with a range from -1
to 1
(bind ?certainty (new FuzzyVariable "Certainty" -1 1));

; add a term to the Certainty FuzzyVariable for Highly Certain,
when the value is between 0.7 and 1
(?certainty addTerm "Highly Certain" (new SFuzzySet 0.7 1));
; add a term to the Certainty FuzzyVariable for Highly
Uncertain, when the value is between -1 and -0.7
(?certainty addTerm "Highly Uncertain" (new SFuzzySet -1 -0.7));
; add a term to the Certainty FuzzyVariable for Certain, when
the value is between 0.2 and 0.7, peaking at 0.45
(?certainty addTerm "Certain" (new TriangleFuzzySet 0.2 0.45
0.7));
; add a term to the Certainty FuzzyVariable for Uncertain, when
the value is between -0.7 and -0.2, peaking at -0.35
(?certainty addTerm "Uncertain" (new TriangleFuzzySet -0.7 -0.35
-0.2));
; add a term to the Certainty FuzzyVariable for Unsure, when the
value is between -0.2 and 0.2, peaking at 0
(?certainty addTerm "Unsure" (new TriangleFuzzySet -0.2 0 0.2));

(defrule fuzzy-enterobacteriaceae-classification "A FuzzyJess
version of the enterobacteriaceae classification rule"
  (observable (name strain) (value GramNegative) (certainty-
factor ?scf))
  ; fuzzy-match returns true if the value of ?scf maps to the
Highly Certain term for the Certainty FuzzyVariable
  (fuzzy-match ?scf "Highly Certain")
  (observable (name morphology) (value rod) (certainty-
factor ?mcf))
  ; fuzzy-match returns true if the value of ?mcf maps to the
Highly Certain term for the Certainty FuzzyVariable
  (fuzzy-match ?mcf "Highly Certain")
  (observable (name aerobicity) (value aerobic) (certainty-
factor ?acf))
  ; fuzzy-match returns true if the value of ?acf maps to the
Certain term for the Certainty FuzzyVariable
  (fuzzy-match ?acf "Certain")
  =>
  ; assert the classification of enterobacteriaceae by
defuzzification using the Highly Certain term
  (assert (classification (class-name enterobacteriaceae)
(certainty-factor (new FuzzyValue ?certainty "Highly Certain") )
)
)
)
```

Figure 5.1: An example showing how the enterobacteriaceae classification rule could be written using FuzzyJess.

6. Comparing Approaches

FuzzyJess requires the user to define FuzzyVariables for every uncertain concept, along with a series of FuzzyValues which describe the various fuzzy concepts related to a FuzzyVariable, by defining a numeric range and natural language term for each FuzzyValue. It also requires that the rules specify which FuzzyValues the uncertainty associated with a piece of data (a fact) must match (as determined by the fuzzy-match function), in order to provide some conclusion. As we discuss in section 5, this has two disadvantages: firstly it means that the developer has to be familiar with Java and the FuzzyJess Java API, as they are required to use both within the Jess rules; and secondly rules only match very specific set of facts, and to cover a range of variable values, many rules have to be written. We believe these features reduce the maintainability of the FuzzyJess KBS; ideally a domain expert should be able to provide the system with the domain knowledge: a task we believe is made significantly more difficult if they have to specify fuzzy variables and values associated with the data. Further, the sizable number of rules required if the system should always produce a conclusion for a given set of facts (regardless of the uncertainty associated with them) also reduces maintainability. For these reasons, we believe our MYCIN-based approach to handling uncertainty in Jess is more practical.

Using the Uncertainty Jess system we have described, it is considerably less complex for the domain expert to specify uncertainties associated with both data and rules. Additionally the procedures needed to calculate combined uncertainties, as we have seen in section 4, are considerably simpler and more succinct than the various counterparts in FuzzyJess.

Essentially, the MYCIN-like approach involves the addition of one extra slot in the deftemplates used in a KBS, to represent the data CF, and the addition of a CF to each rule. In the most extensive application developed to date with Uncertainty Jess, a Haematology Diagnostic Assistant, [5], a Jess system (inference engine & rule set) was linked to a Java front-end program whose GUI presents the user with certainty scales to obtain the data's CF; the same GUI is effectively also used to acquire a CF for each rule from the expert. Once the combined certainty value for a particular conclusion has been calculated, the Java front-end system transforms this into a textual representation, e.g. "Highly Certain" or "Moderately Certain", etc., and the result and the CF are reported to the user.

From the perspective of a domain expert using the above system, the following steps occur:

1. The user inputs the degree of certainty associated with each of their data items.
2. The CF value of all the derived conclusions are calculated.
3. If there are multiple conclusions, the combined CF value is calculated.
4. The final CF value is transformed into a textual representation.
5. The conclusion and its CF are displayed to the user.

7. Conclusions / Future Work

The Uncertainty Jess system has so far only been used with one significant application. We are confident that it is a general approach, but we are now anxious that it is used extensively, so we can get much more feedback. For this reason we are making the following available for download:

1. the Uncertainty Jess rules for the main example used in this paper, i.e. the enterobacteriaceae rule; (deterministic and uncertain versions).
2. the Jess rules and functions needed to compute combined CFs, i.e. the deffunction: *calculate-combined-uncertainty*.
3. the Java classes to acquire uncertainty information from the domain expert and to report uncertainty information to the user (part of the Java front-end system).

This information can be downloaded from <http://www.csd.abdn.ac.uk/~dcorsar/software/UncertaintyJess/>

References

1. B.G. Buchanan and E.H. Shortliffe, editors. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, MA., 1984.
2. CLIPS <http://www.ghg.net/clips/CLIPS.html>
3. E. Friedman-Hill. *Jess in Action: Rule-Based Systems in Java*. Manning Publications Co., Greenwich, CT, 2003.
4. F. Hayes-Roth, D.A. Waterman, and D. B. Lenat. *Building Expert Systems*. Publisher: Addison-Wesley, Reading, MA., 1983. pp. 444
5. A. McKenzie. *Identify: The Haematology Diagnostic Assistant*. BSc. Dissertation, Department of Computing Science, The University of Aberdeen, Aberdeen, 2006.
6. B. Orchard. *The FuzzyJ Toolkit*. http://www.iit.nrc.ca/IR_public/fuzzy/fuzzyJToolkit2.html
7. Protégé <http://protege.stanford.edu>