SENSITIVITY ANALYSIS OF EXPERT SYSTEMS

Wm. Randolph Franklin
Rahul Bansal
Elissa Gilbert

Electrical, Computer, and Systems Engineering Dept.
Rensselaer Polytechnic Institute
Troy, NY, 12180, USA

(518) 276-6077
Internet: Franklin@ Turing.CS.RPI.EDU
Telex: 6716050 RPI TROU
Fax: (518) 276-6003

1. ABSTRACT

The Expert System Parsing Environment has several tools to analyze the sensitivity in expert systems written in IBM's Expert Systems Environment. For each pair of values of each output parameter, we determine all input parameters that have different values for the two elements of the pair. In one example, comparing all pairs chosen from 34 output values of an expert system to determine all the input differences took only 5 minutes in an IBM 4341-12. The numbers of such input differences induce a metric on the output values. We also determine the input parameters that remain unchanged. We calculate the output parameters that are changed when any input parameter changes value. We can also count the number of paths of any length between any two parameter-value pairs or rules, and plot some of these relationships. This system, written largely in Prolog, operates by abstracting a directed acyclic graph from the expert system and then tracing the flow of information through it. The concepts can be extended to any other rule based expert system.

2. INTRODUCTION

An expert system is an expensive term for a program that is written in an expert system shell, which is a programming language, and often a badly designed one from a software engineering viewpoint. Many shells lack concepts of information hiding and modularity. Their mix of forward reasoning, backward reasoning, fuzziness, etc reminds one of Pl/I, which had so many overlapping features that their interactions were not humanly predictable in advance. For example, the Pl/I statement *X = 25 + 1 / 3;* would set *X* to 5.333333 and raise an fixed overflow condition that was usually disabled. As larger expert systems are written, we may expect similar counterintuitive results, and therefore there is a need for systems to debug and test expert systems.

3. SOFTWARE ENGINEERING REVIEW

Since, as expert systems get larger, issues of debugging and testing will become more important, it is reasonable to review past experience with the software engineering life cycle. Experience with large projects has shown that the first four stages — requirements analysis, specification, design, and coding — together consume less than 25% of the system's total cost and time. Debugging and testing require another 25%, while maintenance after delivery costs as much as everything before combined.

However, much current research in expert systems is concerned with issues such as expert system shell, that is language, design. This research, which affects only the coding part of the life cycle process, is about where software engineering was in the 1960s. Some of the few references on debugging expert systems or on graphical access to expert systems are [3,8,9,11]. There are many references to expert systems that debug hardware, such as [2,5]. Prolog has been used to implement expert systems, as described in [4].

An earlier report of the project is [6]. Some relevant theses at RPI are [1,7,10].

4. SYSTEM OVERVIEW

The Expert Systems Environment (ESE) contains two components, the Expert System Development Environment (ESDE), which builds a rule-based expert system, and the Expert System Consultation Environment (ESCE), which consults it. Although it prefers to use an internal format to write expert system files, ESDE can write an expert system in an EXPORT format. Figure 1 shows some sample rules in the EXPORT file format for a toy animal identification expert system, and for a larger expert system used to debug hardware.

```
@RULE GIRAFFE2
@PROP Rule text
if animal_class is 'mammal' and
animal_appearance is 'has long neck' and
animal_color is 'dark spots'
then animal = 'giraffe'
```

```
@RULE RULE123
@PROP Rule text
if crit_volt is "no"
 and kl is "no"
 and thermal is "yes"
 and therm_ind_flashing is "no"
 and (imlp_ind is "no" or
      imlp_ind is "yes")
 and IPC_PC_pon_pb_imm_fail is "yes"
 and cb2le is "no"
 and sr_or_agate_therm_fault is "yes"
then fault is "card A1D2. Replace it"
```

Figure 1: Sample ESE Rules in EXPORTED Format.

We present a graphical overview of our system in figure 2. In Prolog, both data and program files appear similar; we assume that data files contain simple facts, while programs contain procedures with variables. The system names we use here are descriptive rather than the actual file names. PARSER PROLOG reads the EXPORTED file and analyzes the text of the rules to abstract a directed acyclic graph (dag) from the expert system. The nodes of the dag are parameter-value pairs and rules. If a rule tests whether a certain parameter has a certain value then there is an arc from that parameter-value pair to that rule. Likewise, if a

rule sets a parameter to some value, then there is an arc from that rule to that parameter-value pair. PARSER finally writes a Prolog file containing facts describing the nodes and arcs, and some other bookkeeping information such as the expert system name.

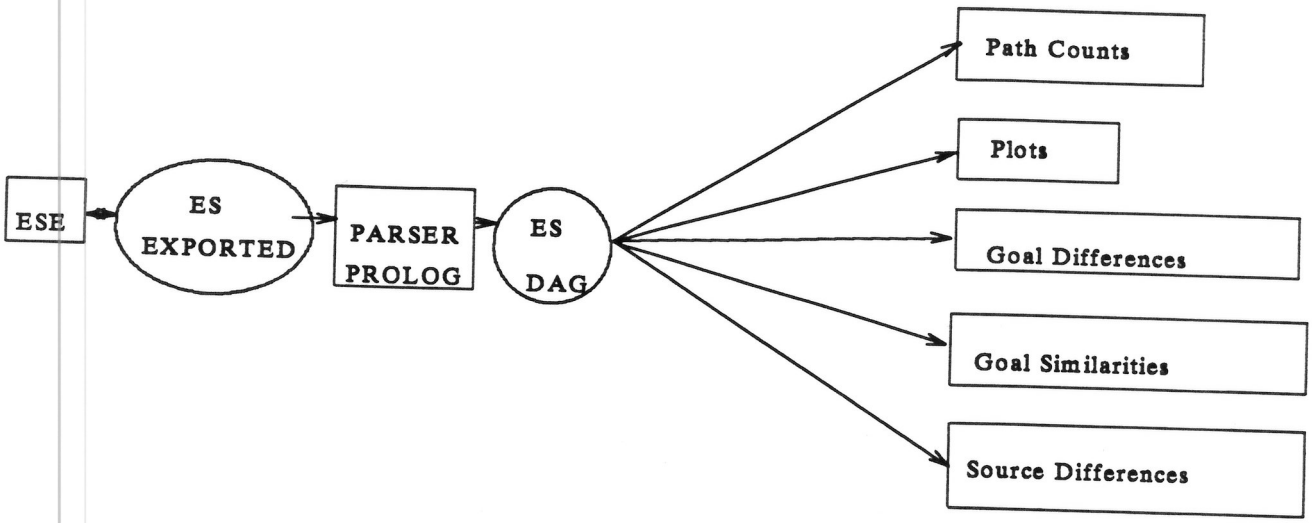

Figure 2: System Overview

A feature of parser is the determination of input parameters (those not written by any rule), and output parameters (those not read by any rule). This is useful for error checking against the user's opinion of what the input and output parameters should be.

This abstraction of the expert system does ignore certain information such as conjunctions and disjunctions within each rule, but does capture a much of the important information about the expert system in a much smaller volume than the original expert system.

The DAG file is read by all the other components of our system; there is no need to refer to the original EXPORTED file again.

Our system contains five components that produce output: path counts, plots, goal differences analysis, goal similarity analysis, and source analysis.

5. PATH COUNTS ANALYSIS

Here the program COUNT PROLOG reads ES DAG and counts the number of paths, of any length, between every pair of nodes in the dag. Although a simple declarative Prolog program count be written for this task, it would require exponential time, so we used a procedural program that required linear time in the total number of paths. Next a PL/1 program, COUNT PLIOPT, formats the table for printing by deleting empty rows and columns and splitting the table into strips if necessary. PL/1 was used because of its powerful formatting capabilities. The two programs communicate through a file, COUNT DATA.

Although a table of path counts bears a superficial resemblence to a low level core dump, it is actually the result of considerable processing. By examining it, one may notice, for example that some value of a parameter have many more paths leading from them than other values of the same parameter. This is an indication of the relative importance that the knowledge engineer placed on those two values.

6. PLOTTING

The plot subsystem, uses several Prolog programs in PLOT PROLOG to calculate the layout of plots and write that information to a file, PLOT DATA. Then a Fortran program reads the data file and produces the actual plot on any of several output devices. Fortran was chosen because of its easy interface to the library plotting routines. We can produce the following plots [12, 7, 10].

- The nodes in a given focus control block (FCB).
- The nodes connected to a given node.
- The tree of focus control blocks.
- The links between only the rules, ignoring parameters.
- An and-or plot of parameters that uses the conjunctions and disjunctions within each rule.

7. GOAL DIFFERENCE ANALYSIS

The subsystem, which is diagramed in figure 3, analyzes what differences in input parameter values can lead to an output parameter being set to one value instead of another. The first program, DIFFS

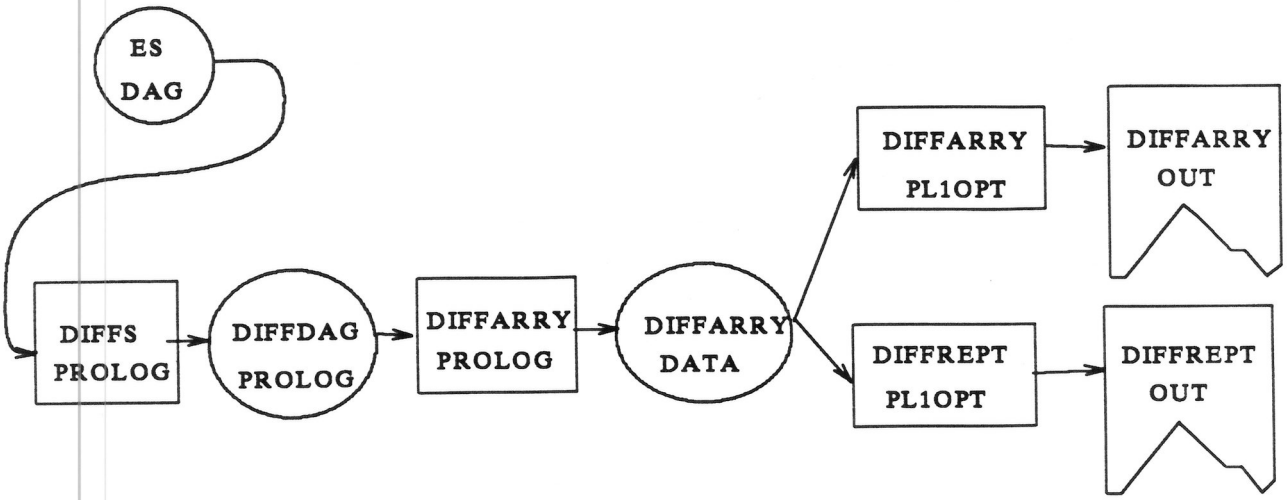

Figure 3: Goal Difference Subsystem

PROLOG, has three options for comparing the values of some specified output parameter.

- Compare two specified values. An example of this is shown in figure 4, where we see that the only difference is that *tiger* is a result of input parameter *animal_color* having value *black stripes*, while *cheetah* results from *animal color* being *black spots*.
- Compare one specified value with all other values.
- Compare all pairs of values. This option can produce voluminous output if there are many values, a file, DIFFDAG PROLOG, is written that other programs can summarize.

DIFFARRY PROLOG reads DIFFDAG PROLOG and writes summary information into DIFFARRY DATA, which can be read by DIFFARRY PLIOPT which writes DIFFARRY OUT. Some sample output from this is shown in figure 5. We see that there are 21 pairs of different values of the output parameter *animal*, and four input parameters, *locomotion*, *appearance*, *class*, and *color*. Output values *albatross* and *penguin* differ in only one of the four input parameters: *locomotion*. However, *albatross* and *giraffe* differ in all four. Thus we have a metric

```
vmprolog
 --- VM/PROLOG 06/04/85 5785-ABH (C) Copyright IBM Corp. 1985 ---
<-consult(diffs).
 GOAL :  <- CONSULT(DIFFS) .
1047MS SUCCESS
 <- CONSULT(DIFFS) .
<-PAIRDIFF.
 GOAL :  <- PAIRDIFF .

PARAMETER NAME?
animal

FIRST VALUE?
'tiger'

SECOND VALUE?
'cheetah'

PARAMETER ANIMAL VALUES 'tiger' AND 'cheetah' ARE DISTINGUISHED BY:

(1) ANIMAL_COLOR = black stripes
(2) ANIMAL_COLOR = dark spots
50MS SUCCESS
 <- PAIRDIFF() .
<-fin.
 GOAL :  <- FIN .
```

Figure 4: Comparing Values *tiger* and *cheetah* of Parameter *animal*

induced on the output values, as shown in figure 6.

Another program, DIFFREPT PLIOPT, can process DIFFARRY DATA to produce DIFFREPT OUT, which summarizes DIFFARRY OUT, as shown in figure 7. Here we see the extreme values. Of the input parameters, *appearance* is the most important, distinguishing 18 of the 21 output value pairs, while *locomotion* is the least important, distinguishing only 11 pairs. Therefore, if questions were equally expensive, we might ask about *appearance* first.

The goal difference analysis has been tested on an expert system with 34 output values. Determining which input parameters distinguished among each half of the 561 pairs took only 5 minutes of CPU time on an IBM 4341-12 using VM-Prolog. The execution time appears

```
VALUE PAIRS:
     1 albatross     penguin          12 ostrich     zebra
     2 albatross     ostrich          13 ostrich     giraffe
     3 albatross     zebra            14 ostrich     tiger
     4 albatross     giraffe          15 ostrich     cheetah
     5 albatross     tiger            16 zebra     giraffe
     6 albatross     cheetah          17 zebra     tiger
     7 penguin     ostrich            18 zebra     cheetah
     8 penguin     zebra              19 giraffe     tiger
     9 penguin     giraffe            20 giraffe     cheetah
    10 penguin     tiger              21 tiger     cheetah
    11 penguin     cheetah


PARAMETERS:
    1 ANIMAL_LOCOMOTION
    2 ANIMAL_APPEARANCE
    3 ANIMAL_CLASS
    4 ANIMAL_COLOR


LAST COLUMN INDICATES NUMBER VALUE PAIRS THIS PARAMETER DISTINGUISHES
  BETWEEN
LAST ROW INDICATES NUMBER PARAMETERS DISTINGUISH BETWEEN EACH VALUE
  PAIR
     1 2 3 4 5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 TOTAL
     ---------------------------------------------------------
   1 * * * * *  *  *  *  *  *  *                                 11
   2   * * * *  *  *  *  *  *  *  *     *  *  *  *  *  *  *     18
   3     * * *  *     *  *  *  *  *  *  *  *                     12
   4     * * *  *     *  *  *  *  *  *  *  *  *  *  *  *     *  17

 TOT:1 2 4 4 4  4  2  4  4  4  4  3  2  3  3 2  2  2  1  1
```

Figure 5: Goal Value Difference Array

linear in the number of pairs, or quadratic in the number of values. Therefore the Expert System Parsing Environment can be used on reasonably sized expert systems.

8. GOAL SIMILARITY ANALYSIS

This subsystem, reports similarities in the input parameters for pairs of values of the output parameter. Two specific values may be compared, one value may be compared against all the others, or all pairs of

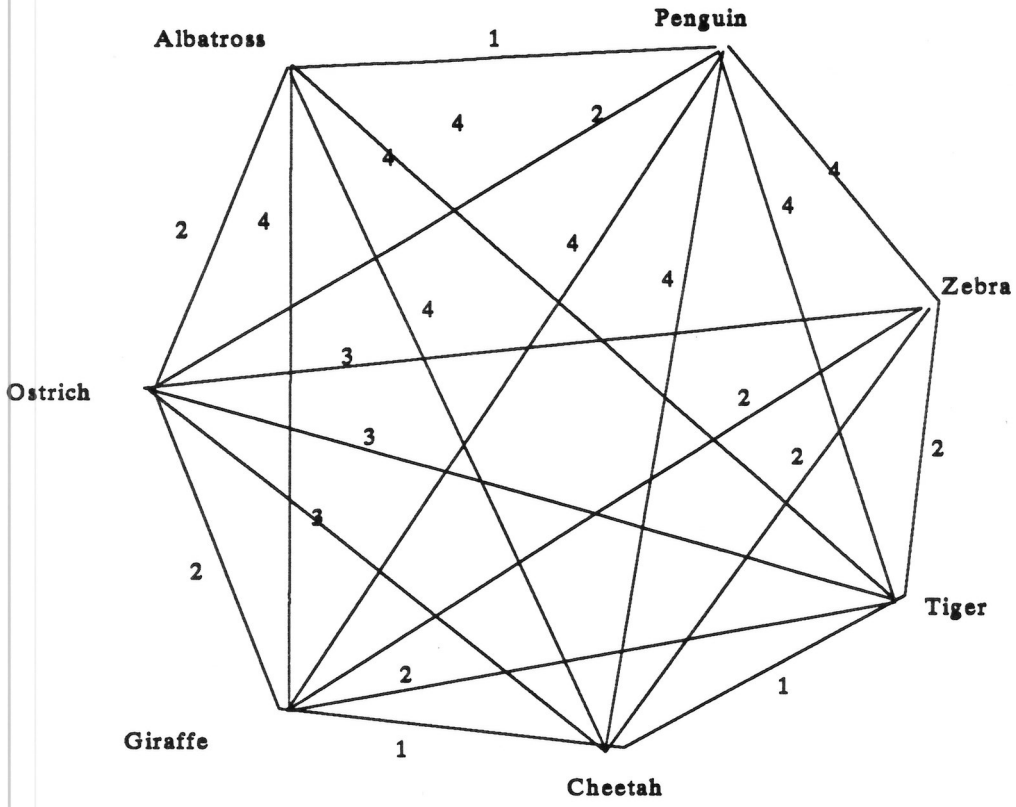

Figure 6: Metric Induced on Values of *Animal*

values may be compared. Figure 8 shows the comparison of *penguin* and *zebra*; they share the same value of *color*, *black and white*, but share no other input parameter values.

The similarity analysis for all pairs of values may be summarized as shown in figure 9.

9. INPUT VALUE ANALYSIS

The previous analyses have been centered on the output parameters. This section reverses the viewpoint to consider which output parameter values can be reached from either or both of a given pair of input parameter values, as shown in figure 10. This compares values *flies* and *flies well* of input parameter *locomotion*. Of all the values of output parameter *animal*, only one, *albatross*, can be reached from both input

```
There were        21 pairs of values,
and          4 parameters make distinctions.

The average number of distinguishing characteristics between goal
value pairs is  2.76


Value pairs with fewer than          2 differences:

albatross, penguin--         1 differences.
giraffe, cheetah--         1 differences.
tiger, cheetah--         1 differences.


Parameters that distinguish more than         15 value pairs:

ANIMAL_APPEARANCE--used for        18 differences.
ANIMAL_COLOR--used for        17 differences.
```

Figure 7: Goal Value Difference Summary Report

```
vmprolog
 --- VM/PROLOG 06/04/85 5785-ABH (C) Copyright IBM Corp. 1985 ---
<-consult(goalsim).
 GOAL :  <- CONSULT(GOALSIM) .
969MS SUCCESS
 <- CONSULT(GOALSIM) .
<-PAIRSIM.
 GOAL :  <- PAIRSIM .
Enter name of goal parameter:
animal
First value?
'penguin'
Second value?
'zebra'
Similarities between ANIMAL = penguin and ANIMAL = zebra:
      ANIMAL_COLOR = black and white

 53MS SUCCESS
```

Figure 8: Goal Similarity Analysis

```
VALUE PAIRS:
    1 albatross     penguin          12 ostrich     zebra
    2 albatross     ostrich          13 ostrich     giraffe
    3 albatross     zebra            14 ostrich     tiger
    4 albatross     giraffe          15 ostrich     cheetah
    5 albatross     tiger            16 zebra     giraffe
    6 albatross     cheetah          17 zebra     tiger
    7 penguin     ostrich            18 zebra     cheetah
    8 penguin     zebra              19 giraffe     tiger
    9 penguin     giraffe            20 giraffe     cheetah
   10 penguin     tiger              21 tiger     cheetah
   11 penguin     cheetah


  PARAMETERS:
      1 ANIMAL_COLOR
      2 ANIMAL_CLASS
      3 ANIMAL_APPEARANCE


        1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
       ---------------------------------------------------------
     1  * * *       * *          *               *       *
     2  * *         *                          *  *  *  *  *  *
     3                                 *                       *
```

Figure 9: Goal Value Similarity Array

```
Comparison of goal ANIMAL values reached by input parameter
ANIMAL_LOCOMOTION values flies and flies well
   Goal values reachable from both input values are:
     ANIMAL = 'albatross'

   Values reachable only by ANIMAL_LOCOMOTION = flies:
   Values reachable only by ANIMAL_LOCOMOTION = flies well:
26MS SUCCESS
```

Figure 10: Input Value Analysis

values, and no output values can be reached from only one. Most output values cannot be reached from either input value.

As before, the data from comparing many pairs of values may be summarized in a table, as shown in figure 11. Here we see a comparison of input values *flies*, *flies well*, and *swims* against output values *albatross* and *penguin*. For example, *albatross* is reachable from *flies* but not from *swims*. *Penguin* is reachable from *swims* but not from *flies*. *Albatross* is reachable from both *flies* and *flies well*.

10. SUMMARY

Programs to determine the sensitivity of the values of the input and output parameters of a rule-based expert system have been described. They form the beginning of software engineering applied to expert systems.

```
Input parameter value pairs:
    1 flies-flies well
    2 flies-swims
    3 flies well-swims


Goal values:
    1 albatross
    2 penguin


Array rows represent input value pairs.
Array columns represent goal values.

Array contents have the following meaning:
   A--the goal value is reachable only from the first input value.
   B--the goal value is reachable only from the second input value.
   C--the goal value is reachable fromboth values in the input pair.

       1  2
     ------
   1   C
   2   A  B
   3   A  B
```

Figure 11: Input Value Analysis Array

11. ACKNOWLEDGEMENTS

This work was supported by the National Science Foundation under PYI grant number CCR-8351942 and by the Data and Federal Systems Divisions of the International Business Machines, Corp.

12. REFERENCES

1. Rahul Bansal, *Debugging, Testing, and Maintaining Expert Systems,* Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, November 1987. M.S. thesis
2. J.S. Bennett and C.R. Hollander, "DART: An Expert System for Computer Fault Diagnosis," *Proc. IJCAI-81*, pp. 843-845, Vancouver, BC.
3. D.C. Berry and D.E. Broadbent, "Expert Systems and Man-Machine Interface - The User Interface," *IEEE Expert Systems*, July 1986.
4. K.L. Clark and F.G. McCable, "PROLOG: A Language for Implementing Expert Systems," *Machine Intelligence*, vol. 10, pp. 455-470, 1982.
5. R. Davis et al, "Diagnosis Based on Structure and Function," *Proc. AAAI Conference*, pp. 137-142, August 1982.
6. Wm. Randolph Franklin, Rahul Bansal, Elissa Gilbert, and Gautam Shroff, "Debugging and Tracing Expert Systems," *Proceeding of the 21st International Hawaii International Conference on System Sciences*, vol. III, pp. 159-167, Kona, Hawaii, January 1988.
7. Elissa Gilbert, *Software Tools for the Maintenance of Expert Systems,* Rensselaer Polytechnic Inst. Electrical, Computer, and Systems Engineering Dept., November 1987. M.S. Thesis
8. D.W. Loveland and M. Valtora, "Detecting Ambiguity: An Example in Knowledge Evaluation," *Proc. IJCAI-83*, pp. 182-184, Karlsruhe, West Germany, August 1983.
9. Mark H. Richer, "An Evaluation of Expert System Development Tools," *IEEE Expert Systems*, July 1986.

10. Gautam Shroff, *EXPLOT: A Software Tool for Analyzing Expert Systems,* Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, 1987. Master's thesis

11. S. Tsuji and E.H. Shortliffe, "Graphical Access to the Knowledge Base of a Medical Consultation System," *Proceedings of American Association For Medical Systems and Informatics Congress*, pp. 551-555, May 1983.