# Exploring the Structure of Rule Based Systems*

**Clifford Grossner, Alun D. Preece, P. Gokul Chander,**
**T. Radhakrishnan** and **Ching Y. Suen**
Computer Science Department, Concordia University
1455 De Maisonneuve Blvd. Ouest
Montréal, Québec, Canada H3G 1M8
cliff@cs.concordia.ca

## Abstract

In order to measure and analyze the performance of rule-based expert systems, it is necessary to explicate the internal structure of their rule bases. Although a number of attempts have been made in the literature to formalize the structure of a rule base using the notion of a rule base execution path, none of these are entirely adequate. This paper reports a new formal definition for the notion of a rule base execution path, which adequately supports both validation and performance analysis of rule-based expert systems. This definition for the execution paths in a rule base has been embodied in a rule base analysis tool called Path Hunter. Path Hunter is used to analyse a rule base consisting of 442 CLIPS rules. In this analysis, the problem of combinatorial explosion, which arises during path enumeration, is controlled due to the manner in which paths are defined. The analysis raises several issues which should be taken into account in the engineering of rule-based systems.

## Introduction and Motivation

Expert systems characteristically achieve a high level of performance in solving ill structured problems [Simon, 1973], using a body of knowledge specific to the problem domain. This knowledge is represented explicitly in the *knowledge base* (KB) of the system, which is kept separate from the mechanism which applies the knowledge to solve problems (the *inference engine*). The intuitive appeal of rules for solving ill structured problems results from rules often being easy for non-programmers to read and write. However, the behaviour of large rule-based systems is almost always hard to predict because, although individual rules can be easy to understand on their own, interactions that can occur between rules are not obvious. As a consequence of this, it is hard to measure and analyze the performance of rule-based expert systems. Tools are required to assist developers in understanding the dependencies that exist between individual rules, and indicate how sets of rules will operate together to complete tasks in the problem-solving process. This paper describes a formal method for detecting the potential interactions between the rules in a rule base [Grossner *et al.*, 1992a], and the development of a tool embodying this method, called Path Hunter [Gokulchander *et al.*, 1992]. Path Hunter is used to analyse the rule base of the Blackbox Expert, an experimental DAI testbed [Grossner *et al.*, 1991]. The Blackbox Expert is a rule-based expert system designed to solve a puzzle called Blackbox.

Our desire to explicate the structure of rule bases is motivated by work in two related fields of artificial intelligence: expert systems, and distributed artificial intelligence (DAI). We use the term *structure* to refer to the dependencies between the rules and potential interactions that can occur between the rules in the rule base. Mapping the structure of a rule-based expert system has become important for structural validation of expert systems [Rushby and Crow, 1990] as well as performance analysis of cooperative distributed problem solving systems (CDPS) [Durfee *et al.*, 1989]. In *structural validation*, the structure of the rule base of an expert system is used as a guide for the generation of test cases and as an indicator of the "completeness" of the validation process [Rushby and Crow, 1990]. Performance analysis of CDPS systems requires a description of the structure of the rule bases of the expert systems in the CDPS to predict the operations that they will be able to perform given the 'information' (data items) available to them as a part of the CDPS system [Grossner *et al.*, 1992b].

We seek to capture the structure of a rule base in terms of chains of inter-dependent rules called *paths*. If the notion of path in a rule base is to be useful for validation and performance analysis of expert systems, it must possess the following criteria:

- The notion of path must be well-defined and unambiguous, so that it can serve as an adequate specification for an automatic path-enumeration program. Only sequences of rules that depend upon each other for their firing are to be considered part of the same path.

*This work was funded in part by Bell Canada Inc.

- When the rules forming a path fire, their combined actions carry out an intended function of the system designer, and can be seen as having significantly advanced the state of the problem being solved.
- The computational effort involved in finding the rules that comprise a path should not be too large to enable efficient automatic enumeration of paths. The number of paths that will be enumerated for a rule base using this definition of path must be computable; that is, we want to prohibit a combinatorial explosion in finding paths.

While researchers in DAI have speculated about the effects of data distribution on the performance of a CDPS system [Lesser and Corkill, 1981; Fox, 1981], there have not been any attempts made at modeling the rule base of an expert system for the purposes of understanding the magnitude of the change in the performance of an expert system given a change in the data distribution of the CDPS. Durfee et al. have observed several of the effects of data distribution on specific test cases of the Distributed Vehicle Monitoring Problem [Durfee *et al.*, 1987]. Other efforts at modeling CDPS systems have been for the purpose of understanding agent behavior and their potential interactions [Shoham, 1991].

The expert system validation literature reports a number of approaches for defining the execution paths in a rule base. The EVA system [Chang *et al.*, 1990] defines a dependency graph (DG) that is used to generate test cases for validating an expert system. The definition of the rule-dependency relation used to construct the DG is unsatisfactory because it allows EVA to consider rules to depend upon each other when in fact they do not; thus, many paths are enumerated which do not reflect 'paths' that will occur when a rule base is executed. Rushby and Crow [Rushby and Crow, 1990] propose a refinement of the EVA DG method, where the rule-dependency relation is improved, but under certain conditions it is still unsatisfactory for the same reason. A stricter method for determining rule dependencies is proposed by Kiper [Kiper, 1992], which models the state of the rule-based system as it would appear when the rules are fired. While this method permits only true rule dependencies to be captured, the rule base states are very costly to compute. Therefore, none of the previous approaches satisfy our criteria.

## The Paths of a Rule Base

Conventionally, an expert system $\mathcal{E}$ is built to solve ill structured problems, which we denote by $P^I$. For our purposes, a rule-based expert system $\mathcal{E}$ is considered to be a triple $\langle E, RB, WM \rangle$ where: $E$ is an inference engine, $RB$ is the rule base used by the inference engine, and $WM$ is the working memory where facts (representing current data) are stored. The facts that are stored in the working memory consists of a predicate name and a list of arguments. Predicates indicate the relationships that exist among the elements of the list in a fact. Let $R$ be the set of all predicates used by $\mathcal{E}$ in solving $P^I$. We use the notation $f_i$ to represent a fact that may be present in the working memory $WM$, where: $f_i = \langle \alpha_i, l_i \rangle$ such that $l_i$ is a list of data elements, and $\alpha_i \in R$ identifies the relationship between the elements of $l_i$. Finally, the *state* of $\mathcal{E}$ is denoted by the set of facts S present in WM at a given time.

The rule base $RB$ of an expert system $\mathcal{E}$ is the set of rules $r_i$ for solving $P^I$. When a rule fires, it changes the state of the expert system by adding or removing facts from the $WM$. We consider a rule $r_i$ to be composed of an LHS and an RHS where: the LHS indicates the fact templates such that at least one instance of each template must be present in $WM$ for the rule to fire, denoted by $\mathcal{I}^{r_i}$; the RHS indicates the set of facts that may be asserted by $r_i$, denoted by $\mathcal{A}^{r_i}$.

The Blackbox expert has been developed using the CLIPS expert system tool [Giarratano and Riley, 1989], and has been designed to solve the Blackbox puzzle. The Blackbox puzzle consists of an opaque square grid (box) with a number of balls hidden in the grid squares. The puzzle solver can fire beams into the box. These beams interact with the balls, allowing the puzzle solver to determine the contents of the box based on the entry and exit points of the beams. The puzzle solver must determine if each location of the grid square is empty or contains a ball, and in addition if the conclusion drawn for the location is certain. As an intermediate step, the puzzle solver can determine that there is evidence indicating that a square is both empty and contains a ball, signalling a conflict. Conflicts may be resolved as additional evidence is obtained. For example, additional evidence may indicate that the ball is certain. Thus, the grid location would be considered to certainly contain a ball, and the evidence suggesting that the location is empty would be disproven. The objective of the Blackbox puzzle-solver is to determine the contents of as many of the grid squares as possible, while minimizing the number of beams fired.

An example rule (`Ball-Certain`) from the CLIPS rule base for the Blackbox Puzzle is shown in Figure 1. Table 1 lists the predicates and user defined functions used by `Ball-Certain` and the other rules from the Blackbox Expert's rule base that will be used for example purposes. The user defined functions represent an indirect access to $WM$; thus, each function will have a predicate associated with it as shown in Table 1. `Ball-Certain` is activated when ample evidence is gathered to support making certain a ball located in the Blackbox grid. This rule will be activated by the presence of the fact using the predicate `BALL_CERTAIN` as well as a fact using the predicate `CERTAIN_BALLS`. Once the rule is activated, it will check to see if the grid location is already certain, in which case no action is needed. Otherwise, the location is made certain; and if a conflict exists, a fact using the predicate `RMC_B` is

```
; Update the grid to indicate that a ball in a particular location is to be considered
: a certain ball.
(defrule Ball-Certain
    ?var1 <- (BALL_CERTAIN ?sn ?rule-ID ?row ?col)   ;A ball is to be made certain
    ?var2 <- (CERTAIN_BALLS ?cb)                     ;Get number of certain balls located
  =>
    (retract ?var1)
    (if (not (iscertain ?row ?col)) then             ;Is the ball already marked as certain?
      (retract ?var2)
      (assert (CERTAIN_BALLS =(+ ?cb 1)))            ;Increment # of certain balls
      (setcertain ?row ?col)                         ;Update the grid making the ball certain
      (if (eq (status ?row ?col) CONFLICT) then      ;Is There a Conflict?
         (assert (RMC_B ?sn ?rule-ID ?row ?col))     ;Indicate the conflict is to be resolved
)))  ; end rule Ball-Certain
```

Figure 1: Sample CLIPS Rule

| USER FN | Interpretation | Assoc. Predicate |
|---|---|---|
| iscertain | check certainty of a square | GMAP_CERT |
| setcertain | set a square certain | GMAP_CERT_B |
| status | check contents of a square | GMAP |

| PREDICATE | Interpretation |
|---|---|
| BALL | Ball located |
| BALL-CERTAIN | A ball is to be made certain |
| BLANK-GRID | Place an empty in a grid square |
| CERTAIN-BALLS | Count of certain balls located |
| CONFLICT_B | Conflict has occurred placing a ball |
| DISPROVE_E | Evidence for an empty square is disproven |
| GMAP | Access to the contents of a grid square |
| GMAP_B | Ball location on the grid |
| GMAP_C | Conflict location on the grid |
| GMAP_CERT | Certainty of grid location |
| GMAP_CERT_B | Ball made certain |
| GRIDSIZE | Dimension of the grid |
| P-BALL | Place a ball on the grid |
| RMC_B | Remove a conflict by placing a ball |
| SHOT-RECORD | exit and entry point for a beam |

Table 1: Predicates and User Defined Function for Blackbox

asserted indicating it can be resolved.

**Ball-Certain** does not follow the form of rules as we have defined them. Therefore, Path Hunter will abstract **Ball-Certain** and split it into two rules: **Ball-Certain%1**, and **Ball-Certain%2**. The rule **Ball-Certain%1** will update the grid to indicate that a ball in a particular location is to be considered a certain ball, a conflict is discovered, and the conflict is to be resolved. $\mathcal{I}^{\texttt{Ball-Certain\%1}} =$ {GMAP, GMAP_CERT, CERTAIN_BALLS, BALL_CERTAIN}, and $\mathcal{A}^{\texttt{Ball-Certain\%1}} =$ {GMAP_CERT_B, RMC_B, CERTAIN_BALLS}. The rule **Ball-Certain%2** will update the grid to indicate that a ball in a particular location is to be considered a certain ball. $\mathcal{I}^{\texttt{Ball-Certain\%2}} =$ {GMAP, GMAP_CERT, CERTAIN_BALLS, BALL_CERTAIN}, and $\mathcal{A}^{\texttt{Ball-Certain\%2}} =$ {GMAP_CERT_B, CERTAIN_BALLS}. The original rule contained a conditional on its RHS representing two different potential actions: the case that the ball made certain was successfully placed, and the case where there was a conflict when the ball was placed. Thus, Path Hunter created two abstract rules each embodying one of the potential actions taken by the RHS of **Ball-Certain**. The predicate associated with the user defined function used in the condition that was on the RHS of **Ball-Certain** has been placed on the LHS of the abstract rules.

In order to reduce the computation and memory requirements needed to solve ill structured problems they are typically decomposed into subproblems [Grossner, 1990], and we denote a subproblem by $SP_t$. Two of the subproblems for Blackbox are **Beam Selection** and **Beam Trace**. Each subproblem $SP_t$ of $P^I$ will have its own set of rules within $RB$. We refer to the collection of rules $\{ r_i \mid r_i$ used to solve $SP_t \}$ as *task* $T_t$. Each subproblem $SP_t$ of $P^I$ will have distinct states that represent acceptable solutions for the subproblem. States which represent an acceptable solution for $SP_t$ are characterized by the presence of facts in $WM$ that use specific predicates called *end predicates*. We denote the set of end predicates for $SP_t$ by $Z_t$.

**Definition 1 (Logical Completion)** *A logical completion for $SP_t$ is a conjunction of selected predicates from $Z_t$ denoting a state which is a meaningful solution to a subproblem.*

We use the notation $SP_t \rightsquigarrow U$ to denote a set of rules $U$ that assert facts using all the predicates of a logical completion for $SP_t$. A logical completion for **Beam Trace** would be GMAP_B $\wedge$ BALL.

We now turn our attention to the types of 'dependency' that can exist between two rules. As the original problem $P_I$ is already decomposed into subproblems, we will only consider dependencies between rules in the same task. Intuitively, we say that one rule is dependent upon another if the action taken by one rule facilitates the other rule to become fireable. The simplest form of dependency exists when one rule asserts a fact that is required by the LHS of another rule. At this point, we will consider only this simple form of dependency.

**Definition 2 (Depends Upon)** *The relation depends upon between two rules $r_i$ and $r_j$ is denoted by $r_i \prec r_j$, and it indicates that the RHS of $r_i$ asserts a fact $\langle \alpha_i, l_i \rangle$ that matches a template in the LHS of $r_j$, with the constraint that $\alpha_i \notin Z_t$. Formally: $r_i \prec r_j \equiv ((\mathcal{A}^{r_i} \cap \mathcal{I}^{r_j}) \neq \emptyset) \wedge (\forall \alpha)((\alpha \in \mathcal{A}^{r_i} \cap \mathcal{I}^{r_j}) \Rightarrow \alpha \notin Z_s)$*

The condition $(\forall \alpha)((\alpha \in \mathcal{A}^{r_i} \cap \mathcal{I}^{r_j}) \Rightarrow \alpha \notin Z_s)$ placed on the *depends upon* relation restricts this relationship to rules that are in the same task.

A set of sample rules from the Blackbox Expert's rule base are shown in Table 2. These rules are part of the `Beam Trace` task. Two examples of the *depends upon* relation exist between **`RA-12-Right%1`**, **`Right-12-Left%1`**, and **`RA-12-Prep%1`**. More precisely, **`RA-12-Right%1`** $\prec$ **`RA-12-Prep%1`** and **`RA-12-Left%1`** $\prec$ **`RA-12-Prep%1`**.

When considering the dependencies that exist among the rules in a task, we become concerned with grouping the rules according to the dependency relationship. Thus for any rule in a task we desire the ability to identify those rules which it *depends upon*.

**Definition 3 (Reachability)** *A rule $r_j$ is reachable from a set of rules $V$, if $V$ contains all the rules that $r_j$ depends upon. We use the notation $V \rightarrow r_j$ to indicate that $r_j$ is reachable from the rules in $V$. Formally, $V \rightarrow r_j$ iff $(\forall r_i \in T_t)(r_i \prec r_j \Rightarrow r_i \in V)$.*

For **`RA-12-Prep%1`** in Table 2, $V =$ {**`RA-12-Right%1`**, **`Right-12-Left%1`**}.

We now consider the set of rules that *enable* a rule to fire. Informally, we say that a set of rules $W$ *enables* a rule $r_j$ when' the rules in $W$ assert facts causing $r_j$ to fire. This set of rules $W$ must satisfy a number of conditions for it to be an *enabling-set* for a rule $r_j$:

- Given $r_j$, then $W \subseteq V$; that is, $r_j$ must *depend upon* every rule in $W$.
- Every rule in $W$ must assert at least one fact that uses a predicate specified by the LHS of $r_j$ where a fact using that predicate is not asserted by any other rule in $W$. Formally, we say $W$ is *minimal* if $(\forall r_i)(\forall r_k)((r_i \neq r_k) \Rightarrow (\mathcal{A}^{r_i} \not\subseteq \mathcal{A}^{r_k}))$, where $r_i, r_k \in W$.
- For each predicate specified by a template on the LHS of $r_j$, if that predicate is used in a fact asserted by at least one rule, then some rule that asserts a fact using the predicate must be a member of $W$. Formally, we say that $W$ is *maximal* if $(\forall \alpha_u)(\alpha_u \in \mathcal{I}^{r_j})((\exists r_k)(r_k \in V \wedge \alpha_u \in \mathcal{A}^{r_k}) \Rightarrow (\exists r_i)(r_i \in W \wedge \alpha_u \in \mathcal{A}^{r_i}))$.

**Definition 4 (Enablement)** *A set of rules $W$ enables a rule $r_j$ iff $W$ is a minimal set of rules that assert facts matching the maximum number of templates in the LHS of $r_j$; we write $W \hookrightarrow r_j$ to denote that $W$ is an enabling-set for $r_j$. Formally, given $r_j \in T_t$ and $V \rightarrow r_j$, $W \hookrightarrow r_j$ iff: $W \subseteq V$, and $W$ is both minimal and maximal.*

For **`RA-12-Prep%1`** in Table 2 there are two enabling-sets: $W_1 =$ {**`RA-12-Right%1`**}, and $W_2 =$ {**`RA-12-Left%1`**}. For **`Remove-Conflict%1`** the enabling-set is {**`Place-Ball%1`**, **`Ball-Certain%1`**}.

A path in a rule base of an expert system must identify a sequence of rule firings that can occur when the expert system is solving a subproblem; thus, each path is composed of a sequence of rules that *depend upon* each other. The set of rules comprising a path must be defined such that each rule in the path is *enabled* by a subset of the set of rules comprising that path. It is desirable that each path represent a 'meaningful' thread of execution for the subproblem; thus, the rules in each path must assert facts using all the predicates of a logical completion for the subproblem.

**Definition 5 (Path)** *$P_k^t$, a path $k$ in task $T_t$ is a partially-ordered set of rules $\langle \Phi, \pi \rangle$ where:*

> $\Phi = \{r_1, r_2 \ldots r_n\}$ *with* $r_i \in T_t$,
> $(\exists U \subseteq \Phi,\ SP_t \rightsquigarrow U)$,
> $(\forall r_i \in \Phi)(\exists W \hookrightarrow r_i,\ W \subset \Phi)$, *and*
> $(\forall r_i \in \Phi)\ (\exists r_j \in \Phi)$ *such that* $[(\forall a_k)(a_k \in \mathcal{A}^{r_i} \Rightarrow (a_k \in \mathcal{I}^{r_j} \vee (a_k \in Z_t)))]$.
>
> $\pi$: *a partial order indicating which rules in path $P_k^t$ depend upon others.*
> $(\forall r_i)((r_i\ \pi\ r_j) \Rightarrow r_i \prec r_j)$.

The condition $(\forall r_i \in \Phi)\ (\exists r_j \in \Phi)$ such that $[(\forall a_k)(a_k \in \mathcal{A}^{r_i} \Rightarrow (a_k \in \mathcal{I}^{r_j} \vee (a_k \in Z_t)))]$ which we have placed on the structure of a path ensures that every fact that is asserted by a rule in a path either uses an end predicate, or it must match the template of the LHS of another rule in the path.

The path formed by the rules in Table 2 as found by Path Hunter is shown in Figure 2. This path represents the combined actions of six rules. These rules recognize a particular configuration on the Blackbox grid, indicate that a ball should be placed on the grid, indicate that the ball is certain, indicate that a location is to be marked as empty, resolves a conflict that occurs when the ball is placed, and disproves that the location should be empty. The logical completion that is asserted by this path is `DISPROVE_E` $\wedge$ `GMAP_C` $\wedge$ `GMAP_B` $\wedge$ `BALL` $\wedge$ `CERTAIN-BALLS` $\wedge$ `GMAP_CERT_B`.

## Experiences with Path Hunter

Path Hunter has been used to analyse the structure of the Blackbox Expert's rule base. This rule base contains 442 CLIPS rules which formed 512 abstract rules. The abstract rules formed 72 equivalence classes (explained below) as well as 170 rules not in any equivalence class, from which Path Hunter found 516 paths. The paths produced by Path Hunter have been verified by the rule base designer as being accurate and

| $r_i$ | $\mathcal{I}^{r_i}$ | $\mathcal{A}^{r_i}$ | *Comment* |
|---|---|---|---|
| RA-12-Right%1 | {GMAP, GRIDSIZE, SHOT-RECORD, BALL} | {RA-12} | Indicate the occurrence of a specific configuration on the Blackbox grid. |
| RA-12-Left%1 | {GMAP, GRIDSIZE, SHOT-RECORD, BALL} | {RA-12} | Indicate the occurrence of a specific configuration on the Blackbox grid. |
| RA-12-Prep%1 | {RA-12} | {P-BALL, BLANK-GRID, BALL_CERTAIN} | Place a ball, mark a location as empty, and indicate that the Ball is a certain ball. |
| Place-Ball%1 | {GMAP, GMAP_CERT, P-BALL} | {CONFLICT_B, GMAP_C} | Update the grid to indicate that placing a ball has created a conflict. |
| Place-Empty%2 | {GMAP, GMAP_CERT, BLANK-GRID} | {DISPROVE_E} | Update the grid to indicate that evidence for an empty grid square, is disproven. |
| Remove-Conflict%1 | {CONFLICT_B, RMC_B} | {GMAP_B, BALL} | A certain ball is placed in a square with a conflict, and the conflict is resolved by placing a ball. |

Table 2: Example Rule Set

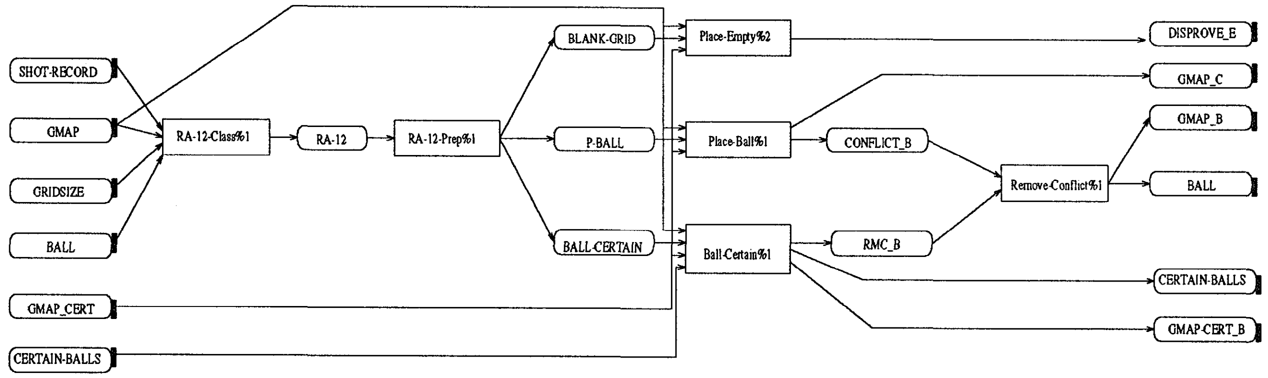

Figure 2: An Example Path

meaningful; that is, they capture the original intent with which the rules in the path were specified and the rules that are depicted in the paths combine together as intended. A typical path contains 5–6 rules, 2–3 branches, and has a length of 4 rules.

The problem of combinatorial explosion will arise when the cardinality of $U$ for many of the rules in a task is large. When the cardinality of $U$ is large, there will be a large number of potential combinations for the rules in a task to form a valid path. Of course, Path Hunter must check all of these combinations. One method that was used to control combinatorial explosion was to form *equivalence classes* of rules. When the rules in a rule base are abstracted, some rules will have the same LHS and RHS. Rules that have the same LHS and RHS are said to form an equivalence class. Rules **RA-12-Right%1** and **RA-12-Left%1**, shown in Table 2, form an equivalence class called **RA-12-Class%1**. The path shown in Figure 2 contains this equivalence class. Thus, the path shown in Figure 2 represents *two* paths that can be observed when the Blackbox Expert's rule base is executed: one path starting with **RA-12-Right%1**, and one path starting with **RA-12-Left%1**. Therefore, equivalence classes reduce the number of paths that must be produced by Path Hunter, with no loss of generality.

Controlling combinatorial explosion can require modifications to the logical completions. When a logical completion is too general, many different paths will be formed that assert this logical completion. Thus, a combinatorial explosion may result. In this case, the rule base designer can control the combinatorial explosion by creating several, more specific, logical completions. This new set of logical completions will lead Path Hunter to create a set of paths for each new logical completion, where the total number of paths for all the new logical completions is less than the paths that were to be created for the original logical completion. In effect, the paths to be created using the original logical completion are broken down into smaller paths where there are fewer potential combinations of rules for creating these smaller paths, resulting in a fewer number of total paths produced. Nevertheless, these smaller paths are still meaningful.

The process of applying Path Hunter to the Blackbox Expert's rule base also served to identify various anomalies: the improper use of predicates, undesired interactions between rules, and rules which were not considered to be part of any path due to programming inconsistencies. In some cases, it was determined that the same predicate had been used within the rule base to reflect slightly different semantics. Thus, it was determined that while the rule base designer had intended to represent two distinct situations, an undetected ambiguity had occurred. These ambiguities also led to undesired potential interactions between the rules in the rule base. One of the rules in the Blackbox Expert's rule base did not appear in any path because it was dependent upon rules in the `Beam Trace` task, but asserted a fact that used an end predicate from the `Beam Selection` task. This situation indicated a poor design for the rule in question. The use of Path Hunter to analyse the Blackbox Expert's rule base also provided a method to validate the design of the rule base by indicating these inconsistencies and ambiguities.

Our experience with Path Hunter points to the need for a well defined approach for the engineering of rule based systems. As the problem to be solved by a rule based system is analysed and a preliminary design for the rule base is created, various issues must be tackled. The modules that will comprise the rule base should be specified as the subproblems that comprise the original problem are understood. The predicates to be used in the construction of the rule base will play a central role in defining the structure of the rule base. It is very important that the semantics attached to each predicate be clear and unambiguous. In addition, predicates must be chosen to ensure that the states which indicate that an acceptable solution to a subproblem are unambiguous. Otherwise, the logical completions will be ambiguous and paths that do not reflect the intent of the rule base designer will be present in the rule base.

## Conclusion

The rule base execution paths defined in this paper meet the requirements for the validation and performance analysis of rule based expert systems. Paths are well-defined because our rule-dependency relations *depends upon* and *enablement* are unambiguous and accurately capture potential rule firing sequences. Paths are meaningful because each path is associated with a logical completion indicating a significant state in the problem-solving process. Paths are computable because the system designer, using logical completions and equivalence classes, can control the complexity of path enumeration.

Path Hunter has been used to analyse the structure of the Blackbox Expert's rule base (512 abstract rules). The use of logical completions and equivalence classes proved effective for controlling combinatorial explosion. Our experience with Path Hunter points to the need for a well-defined approach for the engineering of rule-based systems, where the subproblems (or modules) required to solve the problem, the appropriate solutions for the subproblems, and the predicates to be used in constructing the rule base are clearly specified as early as possible during its development.

## References

Chang, C. L.; Combs, J. B.; and Stachowitz, R. A. 1990. A report on the Expert Systems Validation Associate (EVA). *Expert Systems with Applications (US)* 1(3):217–230.

Durfee, Edmund H.; Lesser, Victor; and Corkill, Daniel D. 1987. Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers* C-36(11):1275–1291.

Durfee, Edmund H.; Lesser, Victor R.; and Corkill, Daniel D. 1989. Trends in cooperative distributed problem solving. *Transactions on Knowledge and Data Engineering* 1(1):63–83.

Fox, Mark S. 1981. An organizational view of distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-11(1):70–80.

Giarratano, J. and Riley, G. 1989. *Expert Systems: Principles & Programming.* PWS-KENT.

Gokulchander, P.; Preece, A.; and Grossner, C. 1992. Path hunter: A tool for finding the paths in a rule based expert system. DAI Technical Report DAI-0592-0012, Concordia University, Montreal Quebec.

Grossner, C.; Lyons, J.; and Radhakrishnan, T. 1991. Validation of an expert system intended for research in distributed artificial intelligence. In *2nd CLIPS Conference, Johnson Space Center.*

Grossner, C.; Gokulchander, P.; and Preece, A. 1992a. On the structure of rule based expert systems. DAI Technical Report DAI-0592-0013, Concordia University, Montreal Quebec.

Grossner, C.; Lyons, J.; and Radhakrishnan, T. 1992b. Towards a tool for design of cooperating expert systems. In *4th International Conference on Tools for Artificial Intelligence.*

Grossner, C. 1990. Ill structured problems. DAI Technical Report DAI-0690-0004, Concordia University, Montreal Quebec.

Kiper, James D. 1992. Structural testing of rule-based expert systems. *ACM Transactions on Software Engineering and Methodology* 1(2):168–187.

Lesser, Victor R. and Corkill, Daniel D. 1981. Functionally accurate, cooperative distributed systems. *IEEE Transactions on Systems, Man and Cybernetics* SMC-11(1):81–96.

Rushby, John and Crow, Judith 1990. Evaluation of an expert system for fault detection, isolation, and recovery in the manned maneuvering unit. NASA Contractor Report CR-187466, SRI International, Menlo Park CA. 93 pages.

Shoham, Yoav 1991. Agent0: A simple agent language and its interpreter. In *Proc. International Conference on Artificial Intelligence (AAAI 91).* 704-709.

Simon, Herbert A. 1973. The structure of ill-structured problems. *Artificial Intelligence* 4:181–201.