National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Canada

Models and Tools for Cooperating Rule-Based Systems

Clifford Grossner

School of Computer Science

McGill University, Montréal,

September 1994

A Thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements of the degree of

Doctor of Philosophy

ISBN 0-612-05716-X

Canada

Models and Tools for Cooperating Rule-Based Systems

To my parents ...

Abstract

A Cooperative Distributed Problem Solving system (CDPS) is composed of a set of agents designed to solve a single problem by working together in a cooperative fashion. In this thesis, we present models and tools which aid the CDPS designer in determining how data items are to be shared between the agents (data distribution) in a CDPS, when the agents are implemented as rule-based systems. Our models consider CDPS from two different perspectives: the inter-agent perspective, and the intra-agent perspective. The inter-agent perspective is concerned with issues relating to the manner in which the agents in a CDPS achieve cooperation. The intra-agent perspective is concerned with the internal structure of an agent, and how an agent is affected when it cooperates with other agents. The inter-agent perspective is modeled using a notion of an "organization" that is imposed on a set of agents sharing data items via a "blackboard"; this model permits the CDPS designer to indicate the mechanism used to achieve both forms of cooperation between the rule-based systems in a CDPS: cooperation by sharing data, and cooperation for control. The intra-agent perspective is modeled using a notion of a "path", which captures the sequence of rules fired by a rule-based system. Based upon the path model, we have developed two tools: Path Hunter and Path Tracer. The models and tools described in this thesis permit the CDPS designer to study the impact of the data distribution chosen for a CDPS on the performance of a rule-based system that will be a member of that CDPS; this helps the CDPS designer in setting the data distribution within a CDPS.

Résumé

Un système CDPS ("Cooperative Distributed Problem Solving") est constitué d' un ensemble d'agents qui ont pour but de résoudre un problème par un travail cooperatif. Dans cette thèse, nous présentons des modèles et des outils qui aide le concepteur de CDPS à déterminer comment les unités de données sont partagées entre les agents (distribution des données) dans un CDPS quand les agents sont réalisés par un système en base de règles. Nos modèles considères les CDPS de deux perspectives différèntes: une perspective inter-agent, et une perspective intra-agent. La perspective inter-agent ce concern des sujets qui traite la manière dont les agents arrive à la cooperation. La perspective intra-agent ce concern de la structure interne d'un agent et de l'impacte sur celui-ci d'entré en cooperation avec d'autres agents. Le modèle de la perspective inter-agent s'appuis sur un concept "d'organisation" qui est imposé sur un ensemble d'agent qui partages des unitées de données par l'intermédiare d'un "tableau" (blackboard); ce modèle permet au concepteur de CDPS d'indiqué le méchanism utilisé pour obtenir les deux formes de cooperation entre les systems en base de règles dans un CDPS: la cooperation par le partage de donnée, et la cooperation pour le control. La perspective intra-agent s'appuis sur le concept de "parcour". Un parcour définit la séquances des règles activées dans le system en base de règles. En s'appuiant sur le modèle "parcour", nous avons développer deux outils: 'Path Hunter' et 'Path Tracer'. Les modèles et les outils décrit dans cette thèse permettent au concepteur de CDPS d'étudier l'impact de la distribution des données choisi pour un CDPS sur la performance d'un système en base de règles qui serra un agent de ce CDPS; ceci aide le concepteur de CDPS à régler la distribution des données à l'intérieur du CDPS.

Acknowledgments

I would like to thank both my supervisors: Thiruvengadam Radhakrishnan and Monty Newborn. Radhakrishnan and I have been working together on different endeavors since 1980, and we have experienced both happy and sad occasions. As I am now at the end of a long journey, I can say that my respect for Radahakrishnan has only increased as I came to know more about him, by being witness to his generosity and maturity. While I have known Monty for a shorter time than Radhakrishnan, Monty has had ample opportunity to demonstrate his patience during the time we have worked together. I am grateful to both my supervisors for all the time and effort they have spent on my behalf.

I would like to thank Alun Preece and Gokul Chander. The three of us spent many hours brainstorming together as we worked out the details of the path model. It was truly a pleasure to work with both Alun and Gokul. I must also thank Alun Preece for the time he took to read my entire thesis and give me his comments. I have no doubt that this thesis is a better product due to Alun's influence. Gokul contributed a significant amount of his time towards the implementation of Path Hunter, and Alun did the same for Path Tracer.

The development of the Blackbox Expert was a large project, and many people made contributions. Kristina Pitula and John Lyons were instrumental in our choice of the Blackbox puzzle as our sample problem. John Lyons made significant contributions to the design of the Blackbox Expert, and Lee Hoc to the implementation and testing of the Blackbox Expert's rule base. Carol De Koven and Kristina Pitula are the Blackbox experts that designed the test set used to validate the functional performance of the Blackbox Expert. Alun Preece was also helpful in the design of the experiment we carried out to validate the functional performance of the Blackbox Expert.

I would like to thank Rick Clark for the hours we spent brainstorming together during the development of the organizational model. Rick spent many hours working on the details of the Consensus protocol. Rick's contribution to our team was instrumental in providing an opportunity for us to begin to interact with other DAI researchers, including Victor Lesser's team at University of Massachusetts.

I would like to thank Larry Thiel for the support he has provided. Larry has been as flexible as possible in permitting me to balance my responsibilities to my full time job and to my research. Larry was also instrumental in arranging for a study leave, granting me the time I needed to study for my comprehensive examination.

I would like to thank Lee Covington. Many theses that I have read contain an acknowledgment to a loved one that has had to be understanding. There is no doubt in my mind that this type of acknowledgment is earned with a lot of hard work on the part of the loved ones referred to by countless researchers. There is also no doubt in my mind that Lee has been an important factor in my success.

Contents

List of Figures

List of Tables

Chapter 1

Cooperating Agents

"The greatest task before civilization at present is to make machines what they ought to be, the slaves, instead of the masters of men."

– Havelock Ellis, "Little Essays of Love and Virtue", 1922

With the advent of high speed communication networks and low cost computing platforms, it has become reasonable to think of many computers, or more precisely programs, working together to solve a large problem. One view taken by researchers in distributed computing is that these programs operate asynchronously, maintain their own data structures, and communicate with each other by passing messages [1]. Researchers in distributed artificial intelligence (DAI) consider these programs to be intelligent agents, each having its own data available locally, and the agents cooperate with each other to solve a problem [2, 3]. The data available locally to an agent is commonly referred to as its *local view* of the problem being solved [3]. The agents are programs constructed by explicitly representing knowledge about how a problem is to be solved.

While researchers in both fields are exploring the issues that arise when multiple programs are used to solve a problem (how to distribute control flow and data flow), the philosophical approach used and the focus of the research carried out in each field is very different. Researchers in distributed computing are concerned with issues such as paradigms for passing messages between programs, detecting global states, programming languages and environments that are appropriate for writing distributed programs, methods for synchronization of programs running on processors with asynchronous clocks, and techniques for debugging distributed programs [4, 5].

Researchers in DAI have been concerned with methods for modeling and programming agents, protocols for cooperation and negotiation among intelligent agents, and appropriate structures for controlling the interaction among intelligent agents [6, 7, 8].

The notion of an agent has been considered by several researchers in artificial intelligence. Minsky proposes that agents are extremely simple entities that can perform actions only as a part of a society of agents [9]. A simple definition of an agent is used by Decker et al.: an agent is an entity that can be considered to have internal states that correspond to belief and intention when observed externally [10]. Rosenschein et al. add the property of rationality when describing the decision making properties of an agent [11]. Shoham considers agents to be entities that follow social laws describing the actions they are permitted to perform [12]. Several models have been proposed by which agents may be programmed [8, 12]. For our purposes, we will consider an agent to be a rule-based system.

In DAI, there are two different perspectives on how agents cooperate. Researchers who consider societies of multiple autonomous agents believe that cooperation occurs when two agents that have distinct problems to solve encounter each other, and the agents determine that they can pool their resources in order to perform some of the tasks that are common to both of their assigned problems; models from game theory have been used show how agents can independently decide that it is advantageous to cooperate [13]. Researchers who view a problem-solving system using multiple agents as a system that is designed with the intention that the agents will cooperate to solve the same problem believe that the agents must be constructed to cooperate with each other; these systems are referred to as cooperative distributed problem solving systems (CDPS) [14]. In both CDPS and distributed systems composed of multiple autonomous agents, the central problem for DAI researchers is how to achieve cooperation among agents, so that the agents can accomplish more as a group than individually. In this thesis, we will be considering only CDPS.

One of the first CDPS constructed was the Hearsay speech recognition system. The designers of Hearsay attempted to achieve cooperation among agents by permitting the agents to share data items that comprised their local view. Hearsay per-

mitted the agents[1] to share the data items in their local view using the blackboard paradigm [15]; a blackboard is a specialized data structure designed specifically for this purpose [16, 17]. Hearsay used a data-driven control mechanism to determine the actions that each agent would perform [15]. In the data-driven approach, each agent signals when it is ready to perform an action. The agents perform their actions opportunistically, exploiting new data as it appears on the blackboard.

While Hearsay did benefit from permitting the agents to share data, experiments with the Hearsay system revealed that the data-driven method can have trouble in converging upon a solution; thus, the agents tend to perform many unfruitful actions. Attempts were made to reduce the number of unnecessary actions performed, giving the system a "focus", by adding a knowledge base and a scheduler to the blackboard [16]. The scheduler used the knowledge base to sequence the actions that the agents were ready to perform. This addition to the data-driven approach provided some improvement, but still could not provide a good focus among the agents. The system could not reason about what data was required to complete the solution it was constructing; hence, it could not trigger the agents to perform actions which would generate the required data. Thus, experiences with Hearsay indicated that simply permitting the sharing of data items among the agents in a CDPS is not sufficient to ensure that the agents can accomplish more as a group than individually.

The Distributed Vehicle Monitoring Testbed (DVMT) was a second generation CDPS system, which included a mechanism to place an ordering on the actions that were performed by the agents [18]. Lesser and Corkill extended the blackboard model used in Hearsay, by adding an agent (referred to as the planner) that was responsible for evaluating the different actions that were proposed by the other agents in DVMT, and by adding a planning section to the blackboard used in DVMT. The planner used the planning section of the blackboard to record data elements that were needed to complete solutions currently under construction. Also, the planner rated the actions that could be performed by the agents to signify how important they might be in solving the problem. Experiments with DVMT have shown that it is beneficial to have a blackboard architecture which combines both an opportunistic and a plan-

[1] Agents were referred to as knowledge sources by Hearsay's developers.

based approach [19]. Thus, in order to achieve cooperation among agents in a CDPS. the system requires a mechanism for evaluating the different actions that can be performed by the agents as well as a mechanism permitting the agents to share data items.

Researchers began to consider various planning protocols, once they understood that achieving cooperation among agents in a CDPS requires a mechanism for evaluating the different actions that can be performed by the agents. Researchers first considered centralized planning protocols where a plan for all agents in the CDPS is constructed by one of the agents [20]. The centralized protocols that have been proposed require that the agent responsible for creating plans maintains a "global view" of the problem being solved, composed from the local view of each of the agents in the CDPS. As the complexity of the problems to be solved increases, the capacity of a single agent to maintain a global view of the problem being solved can be exceeded [20]. This led researchers to consider distributed planning.

Distributed planning protocols do not require a single agent to maintain a global view of the problem being solved. The agents in a CDPS using a distributed planning protocol jointly select the actions to be pursued to solve the problem for which the CDPS is designed. Protocols used for distributed planning allow for three different relationships between the agents; the agents can interact as peers, as in a hierarchy, or as in a free market [7]. Agents that are peers must all agree on the actions they will perform; in a hierarchy, one agent has the ability to issue commands to another, indicating the actions to be performed. Agents that follow the free market model will issue "tenders" for "contracts" upon which other agents will offer "bids" [21]. Distributed protocols should ensure that under a specific set of conditions the agents converge upon an agreement [22].

Cooperation is a notion that has been the subject of much debate. While there is still no universally accepted definition for cooperation, experiences with early attempts at constructing CDPS systems provide us with evidence that cooperation manifests itself in at least two forms between the agents in a CDPS: cooperation by sharing data; and cooperation for control, where agents jointly select the actions to be pursued to solve the problem for which the CDPS is designed. Although construct-

ing CDPS requires mechanisms supporting both forms of cooperation, the issues that must be solved in achieving each form of cooperation are large. Thus, while we briefly touch upon achieving cooperation between agents by having the agents in a CDPS jointly select actions, in this thesis we will focus on achieving cooperation by having the agents in a CDPS share data.

Once researchers had developed mechanisms for permitting the agents in a CDPS to share data items and experimented with various planning protocols in their quest to achieve cooperation among the agents in a CDPS, they began to consider methods for structuring the interaction between the agents. The structure imposed on the interaction between the agents in a CDPS is called an organization [23]. Researchers have speculated that the organization chosen for a CDPS would affect the problem-solving performance of the agents in the CDPS [3, 7, 24, 25]. Experimental evidence from the DVMT testbed indicates that the organization chosen for the agents in DVMT did affect their problem-solving performance [23].

A designer of a CDPS is faced with many choices when selecting an organization: the number of agents to be included in the CDPS, the capabilities that each agent will possess, the planning protocols to be used by the agents, and the manner in which data items are to be shared among the agents in the CDPS (referred to as data distribution). The designer of a CDPS must also ensure that several constraints are not violated when selecting an organization: the processing requirements placed on each agent for problem-solving must not exceed its capacity, and the distribution of data items required for problem-solving must be such that the capacity of the communication channel is not exceeded [7]. The CDPS designer must select an organization that maximizes the performance of the agents within the CDPS.

Various researchers have evaluated the performance of a CDPS in different ways. Lesser et al. consider the *quality* of the results obtained for the problem being solved in terms of the "completeness" of the result and the number of errors it contains, as well as the *quality* (measured by the number of "steps" required to solve the problem) of the strategy by which the results were produced [3]. Other criteria for performance that have been suggested include the computational resources and communication resources consumed to solve the problem [26]. We refer to the quality of the results

obtained and the quality of the strategy used to solve the problem as the *functional* performance of a CDPS, and refer to the resources consumed as *computational* performance. In this thesis, we consider both the functional and computational performance of the agents in a CDPS.

How to construct a CDPS with an appropriate organization for the problem to be solved is still an open question. There is no consensus among researchers as to how the components of an organization can be adjusted to maximize the performance of the agents within the CDPS. While several laboratory prototypes for CDPS have been constructed, there are no models or tools available to the CDPS designer to aid in selecting an organization for a CDPS solving a given problem.

1.1 Thesis Outline

In this thesis, we present models and tools which aid the CDPS designer in selecting an organization for a CDPS. We focus on the use of our models and tools for aiding the CDPS designer in setting the data distribution in a CDPS; thus, cooperation is achieved by having the rule-based systems in a CDPS share data. Our models consider CDPS from two different perspectives: the inter-agent perspective, and the intra-agent perspective. The inter-agent perspective is concerned with issues relating to the manner in which the agents in a CDPS achieve cooperation. The intra-agent perspective is concerned with the internal structure of the agent, and how the agent is affected by the environment in which it must operate — the CDPS. The organizational model (our model for the inter-agent perspective) permits the CDPS designer to specify the mechanism used to achieve both forms of cooperation between the rule-based systems in a CDPS. The path model (our model for the intra-agent perspective) along with Path Hunter and Path Tracer (our tools constructed based upon the path model) permit the CDPS designer to study the impact of the data distribution chosen for a CDPS on the performance of a rule-based system that will be a member of that CDPS, aiding the CDPS designer in setting the data distribution within the CDPS. In this thesis, we focus on mechanisms for setting data distribution, because data distribution affects many aspects of the performance of the rule-based systems in a CDPS [7].

In chapter 2, we present our formal model for the inter-agent perspective, called an organization [27, 28, 29]. Our organizational model permits the CDPS designer to specify the mechanism used to achieve both forms of cooperation between the rule-based systems in a CDPS: cooperation by sharing data, and cooperation for control. Our organizational model also includes a metric for the data distribution of an organization. We then discuss the speculations of various researchers on effects an organization chosen for a CDPS may have on the performance of the agents that are members of that CDPS.

In chapter 3, we discuss the design and validation of our testbed for experimental research [30]. Our testbed is called the Blackbox Expert, and it solves a puzzle called Blackbox. Using the Blackbox Expert, we can conduct experiments in which we change the data items available to a single rule-based system, and we observe its functional and computational performance, simulating the environment the rule-based system would face as a member of a CDPS. We then present results from an experiment that we conducted, using the Blackbox Expert, that quantifies the effects of data distribution on functional and computational performance of the Blackbox Expert solving any Blackbox puzzle [31]. Our experiment also confirms many of the expectations of researchers as to the effects an organization chosen for a CDPS may have on the performance of the agents that are members of that CDPS, when the agents are implemented as rule-based systems [23, 26, 7, 32].

In chapter 4, we describe our model for the internal structure of a rule-based system [33, 34, 35]. Our model for rule-based systems captures the structure of a rule base as chains of inter-dependent rules called paths. We show that our model for the structure of a rule-based system is an improvement over previous attempts by researchers to capture dependencies between the rules in a rule-based system; our model meets three criteria which we believe are essential for modeling the structure of a rule base as sequences of inter-dependent rules, and our model permits the knowledge engineer to control the cost of analyzing the structure of a rule base, a problem which has plagued models proposed by other researchers. We have embodied our formal model for the structure of a rule base into Path Hunter, our tool for analyzing the structure of a rule-base. Using Path Hunter, we analyse the rule-base

of the Blackbox Expert to determine the paths contained in its rule base.

In chapter 5, we discuss our method for applying the path model to study the intra-agent perspective. In our study of the intra-agent perspective, we focus on the relationship between the data items required by a rule-based system and the result produced. We apply the path model to capture the data items required by a rule-based system to produce specific results in two separate steps: paths are analyzed to determine the specific data items required by a rule-based system to achieve each of its goals [36], and then paths are used to monitor the goals that are achieved by a rule-based system as it solves a set of test cases [37, 38, 39]. We present an algorithm for analyzing paths to determine data items required, and a method for monitoring goals achieved. We have constructed Path Tracer, a tool embodying our method for monitoring the goals that are achieved by a rule-based system. We present results from an experiment to validate the accuracy of our method for monitoring goals achieved.

In order to demonstrate the use of our models (organizations and paths) and tools (Path Hunter and Path Tracer), we present a case study for the design of a CDPS to solve Blackbox (also in chapter 5); the agents in the CDPS are constructed using the rule base of the Blackbox Expert. Using the organizational model, we describe a CDPS for solving Blackbox. Then, using the path model, Path Hunter, and Path Tracer, we study the goals that are achieved by the Blackbox Expert as the data items available were reduced. This allows us to establish, for the Blackbox Expert, the relationship between data items available, goals achieved, and result produced. We identify several goals whose achievement is adversely affected by a reduction in the data items available, and using our path model, we are able to determine the specific data items that are responsible for the undesirable behavior that is observed. We then discuss how the CDPS designer can set the data distribution in a CDPS in order to avoid the undesirable behavior that is discovered using Path Hunter and Path Tracer; we present three different scenarios in which the CDPS designer is able to avoid undesirable behavior that was discovered.

It is our strong conviction that theoretical models of sufficient complexity to describe realistic systems require experimental validation, and this view is shared by

Cohen [40]. Models that view agents at a very high level of abstraction, as in game theory (agents modeled by a payoff matrix), require few symbols (representing various variables and parameters of the model), provide a basic set of operators that can be manipulated, and allow for the construction of theorems showing certain properties that are presumably present in the agent [41]. Unfortunately, these models tend to abstract an agent to a great extent. Therefore, it is difficult (if not impossible) to find practical systems containing agents that are sufficiently represented by these simplified models; thus, it is not clear how the results obtained from the simple models can be applied to practical systems. In order for a model to be adequately representative of a practical system, it must contain an adequate level of detail; typically, these models are more detailed than the simple models. These more detailed models will obviously contain many symbols, and may not provide a small set of operators that allow for the construction of theorems, but can still be useful in understanding the properties of a system [40]. In our research, we use models containing sufficient detail to capture properties of agents which are complex rule-based systems. In order to help the reader with the large number of symbols used in our models, we provide a glossary of symbols in Appendix B.

Chapter 2

A Model for CDPS

"Some men see things as they are, and say why?
I dream things that never were, and say 'Why not?' "
– Robert F. Kennedy, quoted in "Esquire", 1969

The interactions that occur between problem solvers cooperating to solve the same problem have been studied by researchers in different fields [42, 32, 23]. In the management sciences, researchers study the interactions that occur when humans cooperate to produce a product; the structure of these interactions is referred to as an organization [32]. In DAI, researchers have been considering the possibility of imposing a structure on the interaction between agents cooperating to solve a problem as members of a CDPS [43, 44]. There exists a number of parallels that can be drawn between the effect of an organization on the performance of humans who are members of that organization, and the effect on performance of a structure imposed upon the interactions that occur between the agents in a CDPS [7].

In this chapter, we will consider a class of problems known as ill-structured problems; ill-structured problems are the type of problems commonly solved by rule-based systems. We will examine the types of organizations that form when humans cooperate to solve problems that are ill-structured. We then present a formal model of an organization for CDPS in which agents are implemented as rule-based systems. We discuss the parallels that are drawn by researchers between the effect of an organization on the performance of humans that are members of that organization and the effects an organization chosen for a set of cooperating agents may have on performance. These parallels identify many of the choices that are faced by the designer

of a CDPS in specifying an organization.

2.1 Ill-Structured Problems

Researchers have developed models for representing problems in a generic manner as well as models for representing specialized classes of problems [45, 46]. Traditionally, the artificial intelligence community has chosen to view problem-solving as a search through a state space [47]. The states in the state space are characterized by a set of *factors* that are extracted from the specification of the problem to be solved. The problem solver will find the solution to a problem by starting from an initial state and then searching through the state space until a state which represents an acceptable solution to the problem has been reached.

We can abstract the state to state transition as a transformation that may be applied to the factors representing that state. The set of all transformations, irrespective of the current state, are referred to as the set of *possible* transformations. Not all the transformations that are possible may be applied by the problem solver at a given state; we refer to the possible transformations that may be applied in a particular state as the set of *legal* transformations.

For our purposes, we will view a problem as a graph with nodes S and edges T:

- S is the set of states in the state space of the problem. The $s_i \in S$ are described by an attribute vector $A =< a_1, a_2, \ldots, a_n >$, and a_i is a "factor" describing a characteristic of the problem state. The set of initial states for the problem is denoted by S^I, and the set of states which are acceptable solutions to the problem is denoted by S^F.
- T is the set of transformations used to traverse the state space of the problem, and T is partitioned into T^L and T^P. T^L is the set of all transformations on s_i that are legal, and $T^P$is the set of all transformations on s_i that are possible, but not legal.

The set of transformations T define a mapping on S; $\alpha : T \times S \rightarrow S$ where $\alpha(t_i, s_j) = s_k$. We say that a state s_n is *reachable* by the problem solver if the state can be reached using only legal transformations. Formally, a state s_n is reachable if there exists a sequence of states $s_1, s_2, \ldots, s_n$ such that $s_1 \in S^I$, $s_{j+1} = \alpha(t_i, s_j)$, $t_i \in T^L$, and $j = 1, \ldots, n-1$. We say that a state is *considerable* by the problem solver if

reaching the state would require the use of at least one transformation that is not legal. Thus, the problem solver may desire to reach a considerable state, but in practice the state cannot be reached. Formally, a state s_n is considerable if there exists a sequence of states $s_1, s_2, \ldots, s_n$ such that $s_1 \in S^I$, $s_{j+1} = \alpha(t_i, s_j)$, at least one $t_i \in T^P$, and $j = 1, \ldots, n-1$. The concept of considerable states was developed primarily for our understanding of ill-structured problems, and it is not developed further in this thesis.

Given the state space model, the problem-solving process that is carried out by the problem solver is as follows: At each state s_i of the problem reached during the problem-solving process, the problem solver must examine the state attribute vector A, and determine the next state to be reached. Typically, the problem solver reasons that the next state to be reached is a final state, or an intermediate state representing a "meaningful" advancement in reaching a final state. The problem solver will then consider all the transformations in T, and using $\alpha(t_i, s_j)$ choose the transformations that are believed to be best suited for reaching the next state. Of course, the problem solver must also be able to determine when a state $s_j \in S^F$ has been reached. The computation that is to be carried out at each state of the problem that is reached must be such that it does not exceed the processing and memory capacity of the problem solver. As $|A|$ and $|T|$ increase, the processing and memory capacity needed by the problem solver also increase.

There have been several informal attempts by researchers to explain the properties that indicate whether a problem is ill-structured [48, 49, 50]. Voss states that in his view ill-structured problems can be characterized as having a large number of open constraints, and the number of constraints is usually much larger than the number of constraints a problem solver can consider [49]. In addition, Voss indicates that when different problem solvers who are considered to be experts in solving a particular problem which is ill-structured are asked to solve the problem, they may produce different results, it is common that each expert will claim that their result is correct, and there exists no known metric for establishing which result is in fact correct. Newell characterized the domain of ill-structured problems as a domain where only weak (heuristic) problem-solving methods were available [50]. Simon states that he

believes that the concept of an ill-structured problem is a residual concept: that is, a concept which is defined in terms of what it is not [48]. In Simon's view, an ill-structured problem is defined as a problem whose structure lacks definition in some respect, and that a problem is an ill-structured problem if it is not a well-structured problem. Simon then goes on to informally describe the properties that he believes can be used to determine if a problem is well-structured (or ill-structured). Given our formalization for the state space model, let us now consider the properties of well-structured problems and ill-structured problems in terms of the parameters of our model.

Well-structured problems: A well-structured problem will have a clear specification of all $a_i \in A$, and a clear specification of the domain of each a_i. The function $\alpha(t_i, s_j)$, the transformations T, and the final states S^F are clearly specified by the problem specification; thus, using $\alpha(t_i, s_j)$ the problem solver only searches the set of reachable states. The $|A|$ and $|T|$ are small compared to the capacity of the problem solver; thus, the computation required at each state does not exceed the capacity of the problem solver.

Ill-structured problems: Ill-structured problems *do not* have as a part of their specification a complete specification for *at least* one of the following:

- the factors a_i that should be included in A to accurately represent the problem state s_i; thus, the complete set of states s_i in the state space S is not known.
- the transformations t_i; thus, the complete set of transformations $t_i \in T$ is not known.
- the state transition function $\alpha(t_i, s_j)$; thus, the problem solver considers searching both reachable and considerable states while solving the problem.
- the final states S^F; thus, two different problem solvers will not always agree on the result they may produce for an ill-structured problem.
- The $|A|$ and $|T|$ are large; thus, solving ill-structured problems requires large amounts of computation and memory resources compared with the capacity of the problem solver.

The solving of ill-structured problems requires the problem solver to approximate the structure of the problem. The problem solver will decide which factors are to

be included in $\hat{A}$, which is an estimate of A. Problem solvers will use their "own knowledge" to approximate $\alpha(t_i, s_j)$, T, and S^F by $\hat{\alpha}(t_i, s_j)$, $\hat{T}$, and $\hat{S}^F$. Problem solvers will choose $\hat{\alpha}(t_i, s_j)$, $\hat{T}$, and $\hat{S}^F$ based on their "expertise", and different problem solvers need not agree [49]. In order to reduce the computation and memory requirements needed to traverse the state space, the problem solver will decompose an ill-structured problem into subproblems. Each subproblem will consider the state of the problem to be represented by $A' \subseteq \hat{A}$, and the set of transformations to be $T' \subseteq \hat{T}$.

We choose to model the problem solver's view of an ill-structured problem as a triple, $P^I = (\hat{P}, P^S, G)$:

$\hat{P}$ is a problem where $\hat{A}$, $\hat{\alpha}(t_i, s_j)$, $\hat{T}$, and $\hat{S}_F$ have been chosen by the problem solver.

P^S is the set of subproblems $P^S = \{SP_1, SP_2, \ldots, SP_m\}$, and each SP_t is characterized by the following:

- PC_t, the set of precedence constraints $PC_t = \{pc_1, pc_2, \ldots, pc_k\}$ for subproblem SP_t. Precedence constraints place conditions on SP_t indicating when the subproblem can be applied to problem $\hat{P}$.
- FPC_t, the set of factors required by the problem solver to decide if all $pc_j \in PC_t$ are met; thus, SP_t is to be applied to $\hat{P}$.
- S'_t, the state space of SP_t, and $S'_t \subseteq S$.
- A'_t, the current state of SP_t, and $A'_t \subseteq \hat{A}$.
- T'_t the set of transformations for SP_t, and $T'_t \subseteq \hat{T}$.

G is the set of goals $G = \{g_1, g_2, \ldots, g_k\}$ used by the problem solver. A goal g_i represents a particular problem state s_i that the problem solver wishes to reach; s_i may or may not be a final state.

The solving of ill-structured problems is typically done by a problem solver using the state space model [48]. Within the state space, the problem solver is able to identify goal states g_i, which are important to reach when solving the problem. Reaching goal states is important because it indicates that a meaningful advancement in solving the problem has occurred. As a mechanism for reducing the complexity of solving ill-structured problems, it is likely that the problem solver will decompose an ill-structured problem $\hat{P}$ into subproblems (SP_t). The precedence constraints for each subproblem PC_t indicate the manner in which subproblems are "related" to each

other with respect to solving the original problem $\hat{P}$. Precedence constraints indicate the sequence in which the subproblems should be solved, and indicate the manner in which the solutions obtained for each individual subproblem can be combined to form a solution for the original problem. In addition, solving ill-structured problems requires a mechanism for handling the fact that the outcome of each step taken by the problem solver cannot be predicted in advance; $\hat{\alpha}(t_i, s_j) \neq \alpha(t_i, s_j)$.

2.1.1 Planning for Ill-structured Problems

Planning involves selecting a sequence of goals to be achieved for solving a problem [51]; this sequence of goals is called a plan. The goals that are selected when planning determine the subproblems that are to be solved by the problem solver, and the subproblems to be solved determine the steps that are to be executed by the problem solver. In the case of ill-structured problems, planning must account for the fact that the outcome of each step taken by the problem solver cannot be predicted in advance; $\hat{\alpha}(t_i, s_j) \neq \alpha(t_i, s_j)$. We now consider how plans can be represented, and consider one method called incremental planning that has been proposed for handling the fact that the outcome of each step taken by the problem solver cannot be predicted in advance [52].

Frames [53] and a partial ordering on goals [54] are two different representations which have been proposed for plans. We consider a plan to be a partially ordered set $P^L = (\gamma, \preceq)$:

- γ is the set of goals chosen by the problem solver when planning for problem P^I, and $\gamma \subseteq G$.
- $\preceq$ indicates the precedence constraints among the $g_i \in \gamma$ indicating an ordering on which the goals are to be achieved.

When planning, the problem solver uses $\hat{\alpha}(t_i, s_j)$ as well as the current and desired value of $\hat{A}$ to create the plan. Each g_k chosen by the problem solver to be in γ will require the problem solver to solve subproblems from P^S. The set of subproblems that need to be solved to achieve g_k is determined by the problem solver using $\hat{\alpha}(t_i, s_j)$, and is denoted by the notation $\beta \Rightarrow^+ g_k$. When choosing g_k, the problem solver will examine the factors required to determine if the precedence constraints for the

subproblems that must be solved to reach g_k are satisfied. Formally, the problem solver will examine $\{a_l \mid \forall SP_t \in \beta,\ \beta \Rightarrow^+ g_k,\ a_l \in FPC_i\}$.

We say that a plan is *complete* for a problem P^I when the least element of the plan is $g_i \in S^I$; and the greatest element is $g_k \in S^F$. A *partial plan* is a plan which is not complete. A plan is said to *fail* when the problem solver executes an SP_t as determined by the goals in the plan and the state s_{j+1} that is reached was not the same as predicted by $\hat{\alpha}(t_i, s_j)$. A plan failure occurs because $\hat{\alpha}(t_i, s_j) \neq \alpha(t_i, s_j)$. With ill-structured problems, the problem solver can determine if a plan fails or succeeds only by executing the plan.

Incremental planning is a method for solving an ill-structured problem P^I where the problem-solving process is an interleaving of planning and execution [52]. With incremental planning, a partial plan is constructed followed by an execution of the SP_t's as indicated by the plan. If the plan fails, then it must be "repaired"; if the plan succeeds, it must be "extended". The repair or extension of a plan requires further planning. Thus, we can view the problem-solving process for an ill-structured problem P^I to be a sequence $PS^I = (PP_1, E_1), (PP_2, E_2), \ldots, (PP_n, E_n)$: PP_i is a partial plan for solving P^I, and E_i is the set of subproblems SP_t executed by the problem solver due to PP_i. PP_1 has a least element $g_k \in S^I$, and PP_n has a greatest element $g_l \in S^F$.

When problem-solving, we assume that each time the problem solver creates the next partial plan PP_{i+1}, the data available are increased. When creating PP_{i+1}, the problem solver will know the new value of $\hat{A}$ that resulted from E_i. The new and previous values for $\hat{A}$, along with $\hat{\alpha}(t_i, s_j)$ are used by the problem solver when planning PP_{i+1}. Thus even if a plan fails, the problem solver may be in a better position to solve the problem because the problem solver is now aware that $\hat{\alpha}(t_i, s_j) \neq \alpha(t_i, s_j)$. The new value for $\hat{A}$ allows the problem solver to repair a failed plan, or extend a successful plan.

The number of partial plans created during problem-solving is affected by $\hat{\alpha}(t_i, s_j)$. A better estimate of $\alpha(t_i, s_j)$ by the problem solver will result in fewer plan failures, and permit more goals to be included in each PP_i. The reduced failure rate of the PP_i's will reduce the interruptions of E_i's for plan repair. Successful PP_i's that

include more goals to be achieved in each E_i reduce the number of PP_i's needed when problem-solving.

Problem-solving using incremental planning is one strategy used for solving ill-structured problems. Incremental planning provides a mechanism by which the problem solver can solve problems even if the outcome of a step taken by the problem solver is not as expected; $\hat{\alpha}(t_i, s_j) \neq \alpha(t_i, s_j)$. When using incremental planning, the problem solver is able to choose one or more steps that seem promising (PP_i), execute those steps (E_i), and then evaluate the state that is reached. One of the optimizations that a problem solver may adopt with incremental planning is to minimize the number of plan failures.

2.2 The Organization

A structure that can be imposed on cooperating rule-based systems solving the same problem is called an organization. Our model for an organization considers the structure of the interaction that can occur between rule-based systems in both phases of problem-solving: planning and execution. In addition, our organizational model requires the CDPS designer to specify the manner in which both forms of cooperation will transpire between the rule-based systems: cooperation by jointly selecting actions, and cooperation by sharing data.

In this section, we first discuss the types of organizations that humans form when cooperating to solve a problem, and consider the factors that contribute to the type of organization that is formed by humans. Then, we present our organizational model, which includes a metric for measuring data distribution. We discuss choices that are faced by the designer of a CDPS, and consider the parallels drawn by researchers in DAI between the effect of an organization on performance in human organizations, and the effects of an organization chosen for a CDPS on the performance of the agents in that CDPS.

2.2.1 Human Organizations

Human organizations provide a mechanism for many people to cooperate with each other to solve problems. Researchers in the field of management science have stud-

ied organizations of people who cooperate to produce products [32, 42]. Human organizations consist of two components called the organizational structure and the coordination structure [26]. The term organizational structure refers to the skills possessed by each person in the organization, and indicates the tasks each person can perform when producing products. The coordination structure specifies the role each person has in decision-making within the organization, and the manner in which information is shared among the people of the organization.

We believe that studying the different organizations formed by humans and the factors which influence the organization chosen by humans will help with the design of organizations for rule-based systems. Many of the factors affecting human organizations are analogous to the factors that must be considered when designing organizations for rule-based systems [7]. An important property of humans, which is one of the factors that determines the organization formed by humans for producing a set of products, is Simon's "principle of bounded rationality" (see [7] page 141). This principle states that the processing capacity of the human brain is limited, as is the amount of information it can assimilate. Thus, the processing demands placed on any person in an organization must not exceed their capacity.

The simplest human organization is the *single person*. One person is responsible for all decisions and skills needed to produce the products. This person must assimilate all the information required both for decision-making and for performing the tasks required to produce the products. The production of most products, except for "simple" products where the processing capacity required for their production is small compared to the capacity of a person, exceeds the processing capacity of a single person.

A *group* is formed when the processing requirements for producing the products increase beyond the capacity of a single person. The members of a group share the tasks required for production, and they participate as peers in decision-making. Each member of the group assimilates all the information required to produce the products. The number of people in the group is based on the number of different skills needed to produce the products. As the size of the group increases, the peer relationship becomes an expensive decision-making method, and the demands made

on the communication channels between the people in the group increases. Peer decision-making becomes more expensive because the processing capacity and time required for the people in the group to arrive at an agreement increase.

A *simple hierarchy* is formed when the processing capacity of the people in a group is exceeded. The simple hierarchy has two levels: the upper level contains a single decision maker who coordinates the tasks performed by the people at the lower level of the organization. The decision maker is responsible for assimilating all information required for deciding which tasks are to be performed by each person on the lower level. Each person is responsible for processing the information required to perform their tasks.

A *uniform hierarchy* evolves when the processing requirements of decision-making increase beyond the capacity of the decision maker in a simple hierarchy. The uniform hierarchy employs multiple levels of decision makers, and the information available for decision-making at each level is an abstraction of the information available to the lower levels of the hierarchy. As the number of levels in the organization and the number of products increase, the allocation of resources among the people performing the tasks required for the production of the products becomes a problem. In this type of organization, the people producing each product compete for the resources available to the organization.

The *multidivision hierarchy* solves the resource allocation problem of the uniform hierarchy by creating a separate uniform hierarchy to produce each product [32]. The separate organizations are controlled by a hierarchy of decision makers, where each organization has its own resources. As with the uniform hierarchy, the multidivision hierarchy uses the technique of abstraction to reduce the processing requirements on the decision makers.

One might argue that the largest human organization is the *free market* [7]. In the free market there are many separate organizations that can produce different products. There is no means of joint decision-making between the different organizations in the free market. The organizations interact by means of "contracts" which are awarded to one organization by another, using a "bidding" system. The organizations in a free market are autonomous; thus there is no guarantee that there will be

an organization willing to accept a contract that has been tendered.

Two factors are believed to be important in determining the type of organization that is formed by humans: the complexity of producing the products, and uncertainty [32]. The complexity of production is the processing required to perform the tasks necessary to produce the products, to assimilate all the information required for production, and the processing requirements for decision-making. Uncertainty is the difference between the information needed for performing a task, or for decision-making, and the information available when the task is performed or a decision is made. Uncertainty affects both decision-making and the tasks performed for production. As the uncertainty with which a person performs a task increases, the quality of results each person produces are reduced: the results produced may not be complete, or the results may contain errors. However, the processing and communication capacity required to perform the task are reduced because there is less information to be assimilated. As the uncertainty with which a decision must be made increases, the processing capacity required by the decision maker increases because certain options will be considered by the decision maker that would otherwise have been quickly eliminated. Decisions taken during conditions of high uncertainty are more likely to be incorrect.

Human organizations employ various methods to cope with the complexity of production. Information complexity is dealt with by using the techniques of *abstraction* and *omission*. When the information complexity for a group exceeds the capacity of the people in the organization, a simple hierarchy is formed where each person is responsible for only a portion of the information required for production. The complexity of performing a task is kept within the capacity of a single person by the continued subdivision of the tasks as they become more complex. In a simple hierarchy, as the complexity of the tasks increases beyond the limits of the people on the lower level of the organization, additional people are added (when considering these organizational models, it is assumed that there is always additional labour available) and the tasks are subdivided. The complexity of decision-making is reduced when subdivisions are created as human organizations evolve from a uniform hierarchy into a multidivision hierarchy.

Uncertainty in human organizations can be handled by allowing slack resources, providing more efficient information distribution facilities to decision makers, and creating peer relationships among the decision makers. Slack resources are extra resources that are used to overcome the problems that occur when incorrect or incomplete results are produced. A more efficient information distribution mechanism and peer relations allow the decision makers to access more information when they take decisions. Relationships between peers in an organization provides a mechanism by which they can share the information at their disposal during decision-making.

The performance of humans in an organization is evaluated by examining the quality of their end products, and the resources they consume [32]. While many factors may be used to evaluate the quality of an end product, two factors that are used to measure the quality of an end product are the product's functionality and reliability [32]. Resources consumed when producing a product include the processing capacity, communication capacity, and the raw materials required for production. The complexity and uncertainty reduction techniques employed by human organizations reduce the amount of resources the humans in the organization consume for producing a given set of products. High coordination complexity and uncertainty during decision-making will result not only in a large amount of processing required for decision-making, but also a poor use of resources.

2.2.2 Organizations for Cooperating Rule-Based Systems

Rule-based systems are typically used to solve ill-structured problems. The rule base approximates T, S^F, and $\alpha(t_i, s_j)$, while the system's working memory contains the current value of $\hat{A}$ [55]. Rule-based systems allow for the fact that $\alpha(t_i, s_j)$ is not known. The techniques for dealing with uncertainty that have been developed for rule-based systems provide a mechanism by which the system will handle $\hat{\alpha}(t_i, s_j) \neq \alpha(t_i, s_j)$ [56]. While problem-solving, a rule-based system using $\hat{A}$ will adjust $\hat{\alpha}(t_i, s_j)$ by updating the confidence placed on the current hypotheses in its working memory.

Cooperating rule-based systems are advantageous for problems where the $|A'|$ and $|T'|$ of the SP_t's are large compared to the capacity of a single system, or where the knowledge bases required for each SP_t are distinct. Knowledge bases are distinct

when they contain expertise from different domains — possibly created by different knowledge engineers. Different methods for representing knowledge can be required to store expertise from different domains [57]. Thus, creating a single knowledge base integrating expertise from different domains is difficult. In some cases, a problem requires the use of rule-based systems that are geographically distributed [58].

In this subsection, we present our model for specifying the mechanisms used to achieve cooperation between the rule-based systems in a CDPS — the organization. Rule-based systems solving a problem P^I can cooperate with each other by sharing data, or by jointly selecting actions. The ability to share data items requires the use of a mechanism for communication between the rule-based systems in a CDPS, and the ability to jointly select actions requires the use of distributed planning protocols. We discuss the communication mechanism and the planning protocols used in the organizational model; then, we present a formal description of the organizational model (including an example). We discuss how the organization selected for a CDPS determines the decision-making responsibility of each rule-based system in the CDPS, and determines the data items required by the rule-based systems in the CDPS when they are planning.

Communication Mechanism: A blackboard is a data structure which allows co-operating rule-based systems to communicate with each other when problem-solving [17, 59]. The data shared by the rule-based systems using a blackboard is organized into levels. A blackboard is a *decomposition hierarchy*, and the data items stored on each level of the blackboard are composed using several data items on the level below it, along with information supplied by the rule-based system. The data items input to the system constitute the lowest level of the blackboard. In some cases, the levels of the blackboard are also an *abstraction hierarchy* [16]. In an abstraction hierarchy, as one goes up in the level of abstraction, the volume of information progressively reduces because details that are no longer required are omitted. This does not consider the value of the information at each level of the blackboard.

A generalization of the blackboard structure for data storage views a blackboard as a hierarchy of levels [60]. Each level is an n-dimensional space where each dimension is an ordered range. The data items are stored on a level of the blackboard using the

dimensions of the level as the index by which they are retrieved. Thus, we have a notion of distance between two data items stored on the same level of a blackboard. When data items on one level of the blackboard are combined to determine the value for a data item on a higher level, the data item could be located at more than one location in the upper level.

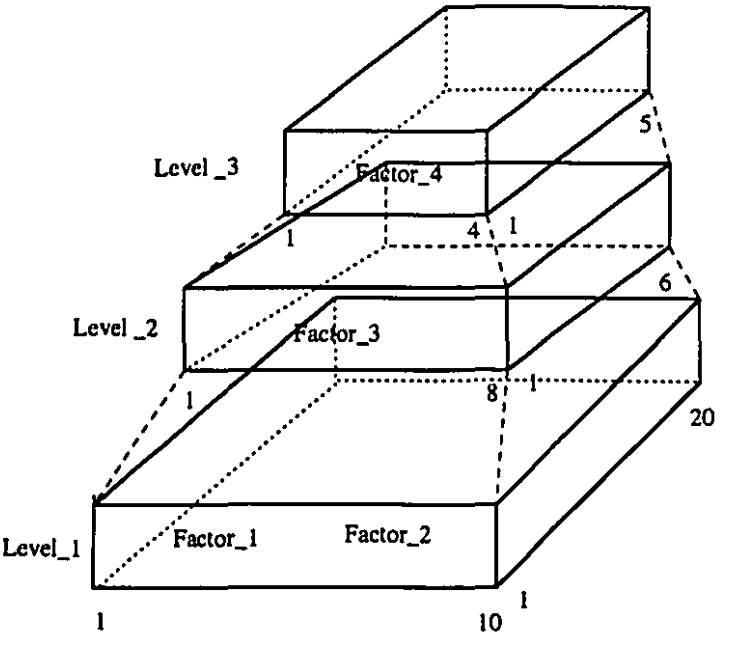
In the organizational model, we view a blackboard, $BB = (LV, \prec)$, as a hierarchical storage structure for $\hat{A}$ of a problem P^l:

- LV is the set of levels for $BB, LV = \{lv_1, lv_2, \ldots, lv_n\}$; lv_i is a blackboard level, $lv_i =< DM, LB, UB, FD >$. DM is the number of dimensions of lv_i, where each dimension is ordered; $LB = \{lb_1, lb_2, \ldots, lb_{DM}\}$ is the set of lower bounds, and lb_j is the lower bound for the range of dimension j of lv_i; UB is the set of upper bounds $UB = \{ub_1, ub_2, \ldots, ub_{DM}\}$, where ub_j is the upper bound for the range of dimension j of lv_i; and FD is the set of factors a_k whose values may be stored on lv_i.
- $\preceq$ is a total ordering on LV indicating the decomposition hierarchy of BB, where lv_i is decomposed into simpler data elements than lv_{i+1}. In the case of an abstraction hierarchy on BB, lv_{i+1} is considered to be at a higher level of abstraction than lv_i.

An example blackboard is shown in Figure 2.1. The blackboard has three levels, and each level has its own dimensions. The factors that are stored on each level are also different, and more than one type of factor can be stored on each level.

A *window* on a blackboard is composed of a number of *regions*, and the window is the mechanism that determines the manner in which the instances of the a_i's stored on the blackboard are distributed among the rule-based systems in a CDPS. A region of a blackboard refers to a portion of one level of the blackboard. We denote a region by $bbr = (lv, LBR, UBR)$: lv is a level of BB; LBR is the set of lower bounds on the dimensions of lv for bbr, and $LBR = \{lbr_1, lbr_2, \ldots, lbr_{DM}\}$ where $lbr_i \geq lb_i$; UBR is the set of upper bounds on the dimensions of lv for bbr, and $UBR = \{ubr_1, ubr_2, \ldots, ubr_{DM}\}$ where $ubr_i \leq ub_i$. A window onto a blackboard is a pair $w = (RG, FC)$: RG is the set of regions $RG = \{bbr_1, bbr_2, \ldots, bbr_n\}$, and $FC = \{A_1'', A_2'', \ldots, A_n''\}$ where $A_j'' \subseteq \hat{A}$ is the set of factors a_i visible in the window when they are stored in region bbr_j.

Each rule-based system will have a window that describes its access privileges for the data items that are stored on the blackboard. When a data item is stored on level



Level_3 = <3, {1, 1}, {4, 5}, {Factor_3}>
Level_2 = <2, {1, 1}, {8, 6}, {Factor_3}>
Level_1 = <2, {1, 1}, {10, 20}, {Factor_1, Factor_2}>

Figure 2.1: Sample Blackboard

lv_j of the blackboard, it will be visible to all the rule-based systems whose window has a region bbr_k that includes the location of the data item on lv_j, and $a_i \in A''_k$ where a_i is the factor whose value is given by the data item. The data items that appear at several locations on lv_j will be visible if any one of the locations at which they are located is in bbr_k.

Planning Protocols: Planning protocols provide a mechanism for avoiding the lack of focus that can occur when rule-based systems are cooperating to solve the same problem [16]. Problem-solving by a set of cooperating rule-based systems, PS^I, requires that they create a partial plan PP_i where they jointly choose γ and then they solve the SP_t's of PP_i. Ideally, S^P should be chosen so that each SP_t can be solved independently. This is generally not possible because of interactions between the SP_t. When problem-solving, the rule-based systems must examine these interactions in choosing γ for PP_i [61]. Planning protocols provide a mechanism by which the rule-based systems jointly choose γ for PP_i.

Planning protocols can be *centralized* or *distributed*, and *binding* or *nonbinding*. A centralized planning protocol has a specialized agent which creates all PP_i. A binding planning protocol obliges the agents to carry out the plan that is created. Early versions of DVMT used a centralized planning scheme [18]. Multiagent Planning [62] and Partial Global Planning [63] are examples of distributed planning protocols. Multiagent Planning is a binding protocol. Partial Global Planning and Contract Net [64] are nonbinding protocols.

In the organizational model, we proposed two binding planning protocols called *Consensus* and *Decree*, which permit cooperating rule-based systems to interact in a peer to peer fashion or hierarchically [27]. The Consensus planning protocol for a set of rule-based systems $PG = \{rbs_1, rbs_2, \ldots, rbs_n\}$, written as $\mathcal{C}\{rbs_1, rbs_2, \ldots, rbs_n\}$, requires that all $rbs_i \in PG$ choose the goals to be included in a plan, and the rbs_i's will examine those instances of $a_k \in \hat{A}$ that are in their respective windows [28, 29]. Consensus is a distributed planning protocol. The Decree planning protocol for a set of rule-based systems $PG = \{rbs_1, rbs_2, \ldots, rbs_n\}$ having rbs_i as the director, written as $\mathcal{D}\{rbs_i : PG - rbs_i\}$, requires that rbs_i chooses the goals to be included in a plan, and rbs_i examines *only* those instances of $a_k \in \hat{A}$ that are in its own window. Decree is a centralized planning protocol. A set of rule-based systems $PG = \{rbs_1, rbs_2, \ldots, rbs_n\}$ that plan using either the Decree $\mathcal{D}\{rbs_i : PG - rbs_i\}$ or Consensus $\mathcal{C}\{rbs_1, rbs_2, \ldots, rbs_n\}$ protocol is called a *planning group*. The *combined window*, denoted by CW, of a planning group is the window used during planning: in the case of Decree, the combined window is the window of the director; in the case of Consensus, the combined window contains "selected" data items from all the windows of the rule-based systems in the planning group. Each rule-based system in the planning group selects data items available in its window to appear in the combined window.

The Organization: The organizational model is used to introduce a structure on the interaction between the rule-based systems in a CDPS [65, 7]. The organization specifies the manner in which both forms of cooperation are to be achieved among the rule-based systems in a CDPS.

Definition 1 (Organization.) *An organization for a CDPS containing m rule-based systems solving problem P^I is a quadruple*

$$\Psi =< ES, CP, CS, WS >$$

where:

ES: the set of rule-based systems in the CDPS; $ES = \{rbs_1, rbs_2, \ldots, rbs_m\}$.

CP: The capability matrix where $cp[i,j] = 1$ *if* rbs_i *can solve* SP_j *else* $cp[i,j] = 0$. *CP is an* $m \times p$ *matrix where* $p = |P_S|$ *and* $\forall j \ \sum_{i=1}^{m} cp[i,j] \geq 1$.

CS: The coordination structure of is a simple connected graph $CS = (ES, ER)$ *with nodes ES and edges ER. The labeled edges ER indicate the planning protocols, Decree or Consensus, to be used by the rule-based systems of the CDPS. CS is subject to the following restrictions:*

- *There is no cycle in CS that contains a Decree protocol.*
- *If a rule-based system is in two planning groups, both planning groups use Decree, the rule-based system is not the director in planning group one, and the rule-based system is the director in planning group two, then the director in planning group one is also directing the rule-based systems in planning group two. Formally,* $\mathcal{D}\{rbs_i : rbs_j\} \wedge \mathcal{D}\{rbs_j : rbs_k\} \Rightarrow \mathcal{D}\{rbs_i : rbs_k\}$.
- *A single rule-based system cannot be directed by two different rule-based systems. Formally, if* $\mathcal{D}\{rbs_i : rbs_j\}$ *then* $\nexists rbs_k \mid \mathcal{D}\{rbs_k : rbs_j\}$.
- *If a rule-based system is in two planning groups, planning group one uses Decree, planning group two uses Consensus, and the rule-based system is not the director in planning group one, then the director in planning group one is also directing the rule-based systems in planning group two. Formally,* $\mathcal{D}\{rbs_i : rbs_j\} \wedge \mathcal{C}\{rbs_j, rbs_k\} \Rightarrow \mathcal{D}\{rbs_i : rbs_k\}$.

WS: The set of windows for ES, and w_i *refers to the window of* rbs_i.

CP indicates the SP_t's that each rule-based system is capable of solving, and CS specifies which planning protocols are to be used by the rbs_i. The Decree and Consensus planning protocols provide hierarchical and peer relationships between the rbs_i. As Decree and Consensus are both binding, they are used to construct coordination structures that are analogous to the types of human organizations discussed in section 2.2.1, except the free market. WS indicates the distribution of instances of the $a_k \in \hat{A}$ (data items) among the rbs_i's when they solve SP_t's, and the CW of each PG indicates the distribution of instances of the $a_k \in \hat{A}$ (data items) during planning.

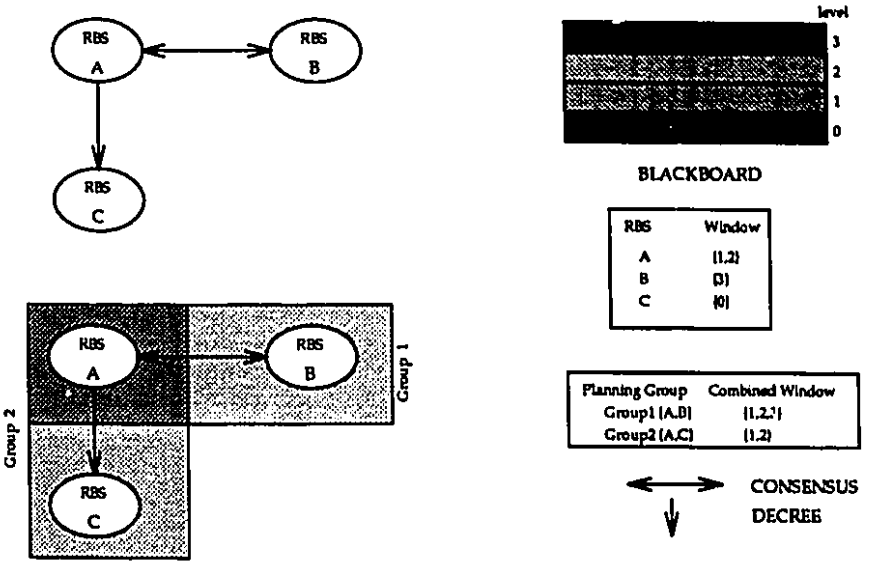


Figure 2.2: Sample Organization for a CDPS

We refer to the distribution of instances of the $a_k \in \hat{A}$ during problem-solving as the data distribution of the CDPS.

A sample organization for a CDPS is shown in Figure 2.2. In this CDPS there are three rule-based systems {A, B, C}, and two planning groups: {A, B,} and {A, C}. Rule-based systems A and B plan using Consensus $\mathcal{C}\{A, B\}$ forming one planning group; rule-based systems A and C plan using Decree $\mathcal{D}\{A : C\}$ forming the second planning group. The window of each rule-based system as well as the combined windows for the planning groups are shown in Figure 2.2.

Decision-Making: The coordination structure is a hierarchy that indicates the division of responsibility for decision-making among the planning groups in the CDPS. The $rbs_i \in PG$ which plan using $\mathcal{C}\{PG\}$ are at the same level in the CS hierarchy. When the $rbs_i \in PG$ plan using $\mathcal{D}\{rbs_j : PG - rbs_j\}$, the director rbs_j is considered to be one level above the $rbs_k \in \{PG - rbs_j\}$. The highest level in the CS hierarchy can be composed of a planning group that plans using Consensus, or a single rule-based

system which is a member of a planning group that plans using Decree: Formally, the highest level of the CS is denoted by either the planning group $\mathcal{C}\{PG_k\}$ if $(\forall rbs_i \in PG_k)(\nexists\, \mathcal{D}\{rbs_j : PG_q - rbs_j\},\ rbs_i \in \{PG_q - rbs_j\})$, or the planning group $\mathcal{D}\{rbs_j : PG_k - rbs_j\}$ where $(\nexists PG_p,\ rbs_j \in PG_p)$. When planning, each rule based system in the CDPS will have its choices for γ restricted. The restrictions for γ are imposed on rbs_i by the rbs_j's on the upper levels of CS.

When the rule-based systems in a CDPS plan, the set of goals $G' \subseteq G$ from which they can select γ is determined by the rule-based systems on the upper levels of CS. The rule-based systems at the highest level of CS do not face any restrictions. The restriction for rule-based systems not on the upper level of CS occurs when a rule-based system rbs_i is a member of two planning groups; once rbs_i has committed to pursue several goals as a member of one planning group, it must choose goals that are "compatible" with its current commitments when planning as a member of the second planning group. Formally, this situation occurs in the following way: Let rbs_i be a member of both planning groups PG_t and PG_r. When the $rbs_i \in PG_t$ plan, they create a plan PP_k. When the $rbs_j \in PG_r$ create a plan PP_l, the $g_s \in \gamma$ of PP_k restrict the choices for γ of PP_l. The goals $g_k \in G'$ from which the $rbs_j \in PG_r$ choose γ must be *compatible* with each $g_s \in \gamma$ of PP_k. A goal g_k is compatible with g_s for an rbs_i if the $\hat{\alpha}(t_i, s_j)$ of rbs_i predicts that there is a sequence of states $s_0, s_1, \ldots, s_z$ where s_0 is the current state of $\hat{A}$, g_k represents the state s_q such that $0 < q < z$, and g_s represents the state s_z.

Data Requirements: We now consider the data items required by the rule-based systems during planning. When planning, the $rbs_i \in PG_t$ use their $\hat{\alpha}(t_i, s_j)$'s, the current value of $\hat{A}$, and G' to create PP_l. For each $g_k \in G'$ chosen by the rbs_i to be in γ of PP_l, the rbs_i will need to examine the $\{a_l \mid \beta \Rightarrow^+ g_k, SP_i \in \beta, a_l \in FPC_i\}$

Definition 2 (Group Planning Set) *The group planning set GPS_t is the set of all factors that can be required by the $rbs_i \in PG_t$ of Ψ when planning:*

$$GPS_t = \bigcup_{(rbs_i \in PG_t)} \begin{cases} (FPC_j) & \text{if } cp[i,j] = 1 \\ \emptyset & \text{otherwise} \end{cases}$$

Summary: An organization is a model for the interactions that occur between cooperating rule-based systems during both phases of problem-solving. An organiza-

tion specifies how the rule-based systems that are members of a CDPS cooperate as follows:

- organizations specify the protocols used by the rule-based systems when planning (CS) as well as the division of responsibility for planning among the rule-based systems. Specifying the planning protocols and the division of responsibility for planning imposes a structure on the interaction between the rule-based systems during the planning phase of problem-solving, specifying how the rule-based systems in a CDPS cooperate by jointly selecting actions.
- organizations specify the availability of data items (a_i) during both planning (CW) and execution (WS). Specifying the window of the rule-based systems (WS) imposes a structure on the interaction between the rule-based systems during the planning and execution phases of problem-solving, specifying how the rule-based systems in a CDPS cooperate by sharing data.

2.2.3 Information Deficit Metric

The information deficit metric is a measure of the data distribution specified when an organization is selected for a CDPS. Figure 2.3 presents an example of the data available to a rule-based system when it solves a subproblem. The figure shows one level of a blackboard depicting the window of a rule-based system as well as the regions where instances of two factors required by the rule-based system to solve a subproblem may be stored. When a rule-based system solves a subproblem SP_t, the information deficit metric measures the "overlap" between the data available to the rule-based system as specified by w_i and the data required by the rule-based system as specified by A'_t. When a rule-based system is planning as a member of a planning group PG_k, the information deficit metric measures the "overlap" between the data available to the rule-based system as specified by CW of PG_k and the data required by the rule-based system as specified by GPS_k.

The information deficit metric that we propose is based on the probability that each instance of an a_j that is required by a rule-based system to solve a subproblem or for planning will be available, as well as the "importance" of each a_j. Let us now consider how we can determine the importance of each a_j, and how we can determine the probability that an instance of a_j is available. Then we will explain the formulation of the information deficit metric itself.

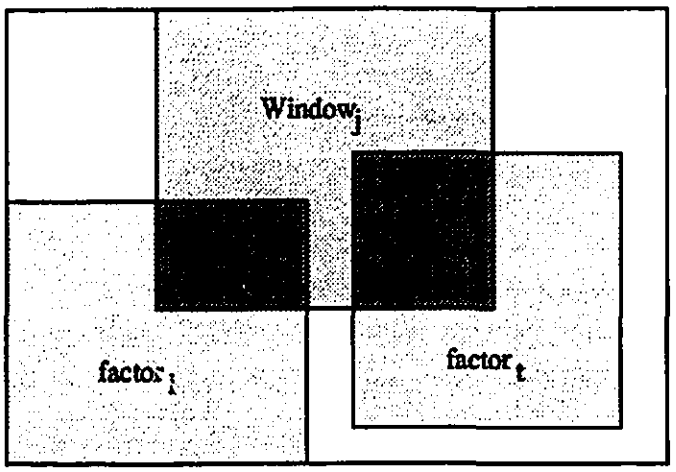


Figure 2.3: Data Items for a Subproblem

Importance of a_j: There are several methods that can be used to establish the importance of each factor a_j required during problem-solving. In our most general formulation of the information deficit metric, we use the *information content* of an a_j to indicate its importance. The information content of an a_j estimates the information gained by a rule-based system rbs_i when it accesses an instance of a_j. In some cases, the problem domain places an ordering on the factors used to represent the state of the problem. In other cases, the structure of the rule-based system used to solve the problem indicates the importance of the various factors that represent the state of the problem.

In the absence of information about the problem domain or the structure of the rule-based systems used to solve a problem, we can make use of a general result from information theory [66]. We can view the blackboard $BB = \{SC_1, SC_2, \ldots, SC_n\}$ as a set of zero memory information sources. The alphabet of symbols $\sum_i$ emitted by SC_i is given by the domain of a_i. This abstraction makes the assumption that the event denoted by a rule-based system rbs_k storing an instance of a_j (a data item) on the blackboard is independent of the event denoted by rbs_l storing an instance of a_k on the blackboard. We also assume that $Domain(a_i)$ is finite and that the probability of each instance of a_i (denoted by v_j) appearing on the blackboard, $P(v_j)$, is known, and $\sum_{v_j \in Domain(a_i)} P(v_j) = 1$. Then the entropy, or the average amount of information

provided by each instance of a_i emitted by SC_i, is given by

$$H(a_i) = \sum_{v_j \in Domain(a_i)} P(v_j) \log_2 \frac{1}{P(v_j)} \text{ bits}$$

Probability that v_j is available: An instance of a_j appears in w_i (is available) when w_i contains a region bbr_k which includes locations on the blackboard where instances of a_j can be stored and $a_j \in A''_k$. The probability that the location where an instance of a_j is stored on the blackboard is in bbr_k is derived from the following:

- the degree of "overlap" between bbr_k and the *fixation* of a_j. The fixation of a_j refers to the bbr_l's in which the instances of a_j can be stored. Formally, the fixation of a factor $a_i \in \hat{A}$ on a blackboard is the set of regions $FX_i = \{bbr_1, bbr_2, \ldots, bbr_n\}$ where the instances of a_i can be stored.
- the expected distribution of the instances of a_j on the blackboard.

The degree of *overlap* between the window of a rule-based system and the fixation of a_i can be measured using the notion of *area* and is denoted by $AREA(FX_i \cap w_j)$. Formally, the area of a set of regions $RG = \{bbr_1, bbr_2, \ldots, bbr_n\}$ of a blackboard is given by

$$AREA(RG) = \sum_{bbr_i \in RG} \prod_{j=1}^{DM_i} (ubr_j - lbr_j)$$

The intersection operator $\cap$ used on a window and a fixation is applied to all the regions in both the window and the fixation, creating a new set of regions. Formally, the operation $\cap$ on FX_i and w_j denotes $\{bbr_t \mid bbr_t = bbr_k \cap bbr_l;\ bbr_k \subset w_j,\ bbr_l \in FX_i,\ a_i \in A''_k\}$. In the example shown in Figure 2.3, the darker shaded regions would be the two regions produced when the intersection operator is applied to $window_j$, $factor_i$, and $factor_j$.

We can now establish a probability for the availability of an instance of a factor a_i in a window w_j: comparing the area of intersection between the fixation of a_i and w_j with the area of the fixation of a_i. If we assume a uniform distribution (of course, if information is available as to the distribution of the instances that are likely to appear for a factor, a different distribution could be used) $\forall v_j \in Domain(a_i)$, then the probability that an instance of a_i stored on the blackboard is in w_j is given by

$$P(a_i/w_j) = \frac{AREA(FX_i \cap w_j)}{AREA(FX_i)}$$

Formulation of the information deficit metric: We now combine the importance of an a_i and the probability that an instance of a_i is available to determine the *information potential* of a_i for a window w_j. We simply use the product of the entropy of a_i and the probability that instances of a_i will be in window w_j to determine the information potential of a_i. Formally, the information-potential of a factor a_i for a window w_j is given by $\mathcal{IP}(a_i/w_j) = H(a_i)P(a_i/w_j)$

Using the entropy and the information potential of the $a_i \in A'_t$ we can derive the information deficit for the rule-based systems in an organization when solving a subproblem, SP_t. The information deficit metric is constructed such that the range in value is between 0 and 1. A value of 1 reflects the situation that no information is available, and a value of 0 indicates no deficit and that all the information required will be available, if it is present on the blackboard. In computing the information deficit, we start by computing a value for the availability of instances of all factors required to solve a subproblem: a sum is taken of the information potential for each factor required to solve the subproblem; then, we divide this sum by the value for availability if all instances of every factor required to solve the subproblem are available (the total entropy of all factors required to solve the subproblem), normalizing the value obtained for the information available. Formally, the information deficit when a rule based system rbs_i is solving a subproblem SP_t is given by the following:

Definition 3 (Execution-Time Deficit) *The information deficit of a rule-based system rbs_i in an organization when it solves SP_t is*

$$DE(rbs_i/SP_t) = 1 - \frac{\sum_{a_k \in A'_t} \mathcal{IP}(a_k/w_i)}{\sum_{a_k \in A'_t} H(a_k)}$$

Using the entropy and information potential of the factors a_i required by a planning group PG_k, we can derive the information deficit for the rule-based systems in an organization when they plan. The same principle used to determine the information deficit when solving a subproblem applies to the information deficit when planning, but for planning we consider the factors required by the planning group and the planning protocol that the planning group uses. Formally, the information deficit when a rule based system rbs_i is planning in planning group PG_k is given by the following:

Definition 4 (Planning-Time Deficit) *The information deficit of a rule-based sys-*

tem rbs_i in an organization when planning as a member of planning group PG_k is

$$DP(rbs_i/PG_k) = \begin{cases} 0 & \mathcal{D}\{rbs_j : PG_k - rbs_j\} \\ 1 - \frac{\sum_{a_l \in GPS_k} IP(a_l/CW_k)}{\sum_{a_l \in GPS_k} H(a_l)} & otherwise \end{cases}$$

The information deficit metric provides a mechanism to measure the availability of data items (instances of a_i) among a set of cooperating rule-based systems during both stages of problem-solving. Using the information deficit metric, we can measure the availability of data items when the rule-based systems in a CDPS jointly select goals, and measure the data items available to each rule-based system in a CDPS as it solves a subproblem.

2.2.4 Choosing an Organization

In DAI, researchers have informally discussed and studied CDPS systems. Researchers have speculated about the effect on performance of an organization that is selected for a CDPS. There exists some empirical evidence in the literature to support the expectations of researchers for certain specific test cases; Durfee et al. have observed the performance of the agents in a CDPS, given the organization selected for specific test cases of the Vehicle Monitoring Problem [23]. Using the organizational model as a formal framework, we now consider the informal speculations of various researchers as to the effect on performance of choices made by the CDPS designer when selecting an organization for a CDPS.

Selecting an organization for a CDPS requires a choice of ES, CS, CP, and WS. Researchers in DAI believe that many of the issues that arise with human organizations are thought to affect the design of organizations for CDPS [7]. The principle of bounded rationality applies both to humans and to the agents in a CDPS [7]. As problems grow more complex, requiring more processing capacity, the limits of the agents will be exceeded. Agents in a CDPS will be faced with complexity issues, as are humans: the agents must perform the necessary operations to solve the required subproblems (SP_i), the agents must process the data items in their windows (w_i), and the agents must decide the subproblems that are to be solved ($\{SP_i \mid \forall g_k \in \gamma,\ \beta \Rightarrow^+ g_k,\ SP_i \in \beta\}$). Uncertainty will also be present in a CDPS: when problem-solving, the agents must solve subproblems and must determine the

subproblems to be solved using only the data items available in their windows.

The performance obtainable with a particular organization will be of interest to the CDPS designer. As with human organizations, performance will be concerned with the quality of the results produced (functional performance) by the agents as well as the resources consumed (computational performance) [23]. The quality of the results produced is typically assessed by using problem dependent measures. The resources consumed would be measured in terms of the processing and communication capacity required to solve the problem. Of course, the CDPS designer wishes to maximize the functional and computational performance of the agents in a CDPS when selecting an organization. Thus, strategies are required for controlling the effects of processing complexity and uncertainty.

Researchers expect that the choices made for ES and CP determine the manner in which the processing load incurred when solving a subproblem is shared among the agents in the CDPS. As the number of agents in the CDPS that possess the capability to solve a subproblem increases, the processing requirements for solving that subproblem placed on each of them is expected to decrease. However, the communication resources consumed by the agents solving a subproblem are expected to increase. Depending on the planning protocol used (Decree or Consensus) between the agents, the cost of planning may also increase.

The use of Decree and Consensus when constructing CS is expected to affect the cost of planning, and as the number of agents in the CDPS increases, the number of different configurations for CS increase rapidly [67]. Agents that plan by Consensus benefit from an increase in the number of data items available during planning. The use of additional data items when planning can result in "better" plans, but the cost of creating the plan increases because we anticipate an increase in communication overhead and processing capacity required to assimilate the additional data items. The cost of planning using Decree is less than that for Consensus because of the increased number of data items that must be processed by each agent when Consensus is used for planning. Thus, the use of Decree at appropriate points in the CS of the organization can reduce the cost of planning.

Changing WS adjusts the data distribution in a CDPS, and is believed to affect

the computational and functional performance of the agents [32, 26, 12, 3, 7, 24, 25]. Researchers have speculated that when an agent solves a subproblem, as the number of data items in its window that are available to solve the subproblem decreases, the following occurs: the number of errors in the results produced for the subproblem increases, the quality of the results produced decreases, and the processing and communication resources consumed by the agent to process the data items required to solve the subproblem decreases [3, 7]. When the agents plan, the effect of the data distribution on the processing and communication resources consumed will depend upon the type of planner that is used. In the case of a planner which synthesizes goals using the data items that are in the combined window of the planning group, or adjusts the rating of a predetermined set of goals using the data items that are in the combined window of the planning group, it is expected that the processing and communication resources consumed by the agents to create the plan decrease as the number of data items available decreases. In the case of a planner that uses the data items that are in the combined window of the planning group to eliminate goals from a predetermined set of potential actions, it is expected that the processing and communication resources consumed by the agents to create the plan increase as the number of data items available decreases; this occurs because many options are explored that could have been quickly eliminated if more data items were available. When planning, as the number of data items that are available in the combined window of a planning group decreases, there may be an increase in the number of plan failures. An increase in the number of plan failures could occur because the agents in the planning group choose goals to be included in the plan that would not be selected if they could access all the data items required for planning.

2.3 Conclusion

Our organizational model provides a formal framework for studying the effect (on performance) of the organization selected for a CDPS in which the agents are implemented as rule-based systems. The organizational model provides a mechanism for specifying how the rule-based systems in a CDPS cooperate; the model includes a formal description of the planning protocols used to define the coordination structure,

the structure of the blackboard used for sharing data items, the availability of data items during both phases of problem-solving, and a metric for the data distribution in the CDPS (based upon a notion of distance between two data items stored on a blackboard). The rule-based systems that are members of a CDPS employ the problem-solving model for ill-structured problems we described in section 2.1.

Design of a CDPS is a complex task, and no single thesis can address all aspects of CDPS design. In this thesis, we will restrict ourselves to consider only the choices that face the CDPS designer in setting the data distribution (WS) in a CDPS, given that ES, CP, and CS have already been specified. Considering the informal speculations by different researchers in light of our formal framework for CDPS systems, we identify two issues which (studied in this thesis) we believe are important to the CDPS designer when setting the data distribution in a CDPS:

- for a CDPS in which the agents are implemented as rule-based systems solving a specific problem, how important is the data distribution in the CDPS in determining performance?
  - we lack evidence that the trends expected by researchers will actually occur in an operational rule-based system.
  - we lack evidence that the effects of data distribution on performance are important for specific problems, or classes of problems, other than the specific test cases reported in the literature.
- how does the availability of a specific data item affect the performance of the rule-base systems in the CDPS?
  - we are unable to determine exactly how to set the availability of specific data items in order to improve the performance of the rule-based systems in a CDPS, or avoid serious failures.

Chapter 3

A Testbed for CDPS

"It is common sense to take a method and try it.
If it fails, admit it frankly and try another.
But above all, try something."
- Franklin D. Roosevelt, Speech, 22 May 1932

Designing a testbed for experimental research requires the consideration of different requirements. Of primary importance is the type of experiments that will be carried out using the testbed; that is, the hypotheses that will be tested in the experiments, the input parameters required for conducting the experiments, and the outputs that are to be measured during the experiments. The method chosen for the implementation of the testbed is also important; the cost of constructing the testbed as well as the costs of conducting the experiments for which the testbed is intended must be justified by the gains that are expected from the experiments. Several of the prototype systems used in DAI research have been very complex, and have required several person-years of effort. For example, the work expended in the DVMT project is estimated to be 15 to 20 person-years.

The design and implementation of a testbed for experimental research is a large task, typically requiring the participation of many people; our testbed is no exception. My role in the design, implementation, and use of our testbed has been to participate in the conception of the design for the testbed, to act as a project leader for the graduate students who worked on the implementation of the testbed, to integrate the individual components of our testbed that were constructed by the graduate students into a single functioning system, to design the experiments to be conducted using

the testbed, and to carry out these experiments. The people who participated in the design of our testbed are as follows: Kristina Pitula and John Lyons were instrumental in choosing the sample problem for our testbed; John Lyons made significant contributions to the design of our testbed; Lee Hoc worked on the implementation and testing of the rule base of our testbed; Carol De Koven and Kristina Pitula are the domain experts for our sample problem, and they designed the test set used to validate the functional performance of our testbed. Alun Preece helped with the design of the experiment we carried out to validate the functional performance of our testbed.

The long term goal of our research is to explore the implications of the different design choices available to the CDPS designer when selecting an organization for a CDPS in which the agents are implemented as rule-based systems. In this thesis (our current research), we will focus on the choices available to the CDPS designer in specifying data distribution (WS). In consideration of both our current research and longer term research goals, we envisage our testbed to be used for different types of experiments, in each stage of our research, as follows:

Stage 1: we expect to conduct experiments in which we can change the data items available (a_i) to a single rule-based system, and observe the functional and computational performance of the system. In stage 1, our testbed consists of a single rule-based system, which allows us to specify its window (w_i) as if it were a member of a CDPS, simulating the environment the rule-based system would face as a member of a CDPS.

Stage 2: we expect to conduct experiments in which we can change the organization imposed on a set of cooperating rule-based systems, while we observe the functional and computational performance of the rule-based systems. In stage 2, our testbed would be a complete CDPS, consisting of several rule-based systems constructed from the original single rule-based system used in stage 1.

In our current research, reported in this thesis, we only conduct experiments of the type described in stage 1. However, we will discuss the features that have been included in our testbed that will allow us to transform our testbed to conduct the type of experiments we expect to perform in stage 2 of our research.

Based upon the experiments that we intend to conduct, our testbed must include the following:

- A sample ill-structured problem that can be decomposed and solved by several cooperating problem solvers. In order to quantitatively measure functional performance, we also require a metric for evaluating the quality of any result produced for the sample problem,
- A rule-based system for solving the sample problem. It must be possible to specify the availability of data items, and it must be possible to measure the computational performance of the system. In addition, we require that the rule-based system be constructed in a manner that will permit its components to be reused when we construct a CDPS for solving our sample problem.
- The construction of any system is not complete until the system has been tested, demonstrating that the system is capable of solving the problem for which it has been constructed. In the case of our testbed, validating its ability to solve our sample problem is especially important, because we intend to study the system's ability to solve our sample problem as we vary the data items available. In order to be certain that our testbed can properly exhibit the reduction in performance that we expect when the number of data items available to the system is reduced, we must establish that the ability of the system to solve our sample problem is "respectable" when all data items are available.

In this chapter, we present the design of our testbed, and present results from an experiment we conducted using our testbed. We describe the sample problem we have chosen for our testbed, which is called Blackbox. We explain why Blackbox is considered to be an ill-structured problem, and discuss the decomposition of Blackbox. We present our rule-based system for solving the Blackbox puzzle, called the Blackbox Expert. We explain how the Blackbox Expert permits the experimenter to control the availability of data items, and how the components of the Blackbox Expert can be reused to construct a set of rule-based systems that will cooperate to solve the Blackbox puzzle. We then discuss the method used to validate the functional performance of the Blackbox Expert; we compare the results produced by the Blackbox Expert for a set of test cases with the results produced by humans for the same set of test cases. We then present results from an experiment that we conducted, using the Blackbox Expert, that quantifies the effects of data distribution on functional and computational performance, confirming the expectations of researchers as discussed in chapter 2.

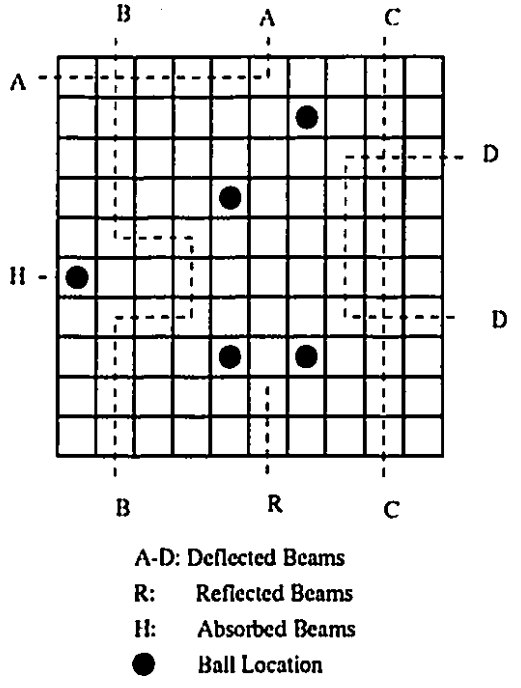


Figure 3.1: Beam Behavior in Blackbox

3.1. 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. As illustrated in Figure 3.1, the beams may be fired from any of the four sides of the box (along one of the grid rows or columns) and follow four simple rules:

- If a beam hits a ball, it is absorbed (labeled by 'H').
- If a beam tries to pass next to a ball, it is deflected 90 degrees away from the ball in the square diagonally next to the ball (labeled alphabetically except for 'H' and 'R').
- If a beam tries to enter the grid at a square adjacent to a border square that contains a ball, it is reflected back 180 degrees (labeled by 'R').
- If a beam tries to pass between two balls, it is reflected back 180 degrees (labeled by 'R').

The objective of the puzzle solver is to determine the contents of as many of the grid squares as possible, while minimizing the total *value* of beams fired. Beams that

are absorbed or reflected have a value of one point, while deflections have a value of two points. Beams that are absorbed or reflected are given one point because the puzzle solver only knows the entry point of the beam, and that the beam did not exit the blackbox. Beams that are deflected are given a value of two points because the puzzle solver knows the entry point of the beam as well as the exit point for the beam; the additional data indicating the exit point of the beam is very useful to the problem solver in determining the contents of the grid squares.

The Blackbox puzzle is solved iteratively by firing beams and observing their exit points from the grid. The information obtained from observing the exit points of the beams, and the problem solver's knowledge of how the beam can be affected by the balls within the box are used to draw conclusions about the contents of the box. In addition, the problem solver must decide if the conclusion drawn for a grid square 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, signaling 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 square would be considered to certainly contain a ball, and the evidence suggesting that the square is empty would be disproven.

In certain configurations of the Blackbox grid, the contents of a number of grid squares cannot be identified; we refer to these as *shielded regions.* A region is called a shielded region if it is a proper subset of the Blackbox square, it contains at least one ball, and the ball is shielded by other balls that are surrounding it, so that no beam can penetrate into the region. An example of a shielded region is shown in Figure 3.2, wherein the shielded region is shaded. No beam can penetrate into the shaded region. The balls contained in this region are called "unmappable balls". In the case of a shielded region, the puzzle solver can only state that the contents of each square in the region remains unknown.

Unlike the case of other well known games and puzzles such as chess, there are no known ways of rating a result produced by a Blackbox puzzle solver. In consultation with a group of Blackbox experts, a metric has been devised to evaluate the quality of a result produced for any test case of the Blackbox puzzle. The factors that

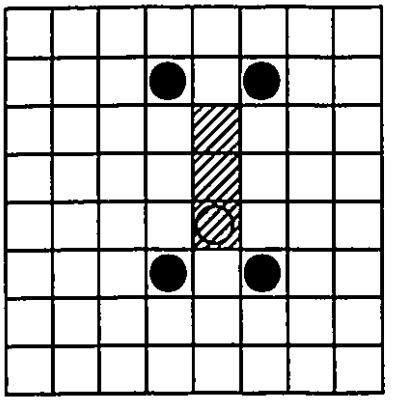


Figure 3.2: An Example of a Shielded Region.

were chosen to determine the quality of a result are: the number of balls that were correctly located, the number of grid squares (other than those which contain balls) whose contents are correctly identified, and the total value of the beams fired to solve the puzzle. As stated in the objectives of Blackbox, the best result would have all the balls and grid squares correctly identified as well as a minimum total value for the beams that were fired.

The metric that was adopted is as follows:

$$SCORE = \left(2 - \frac{B^C - B^W}{B^M} - \frac{L^C - L^W}{L^T} + \frac{b^V}{b^T}\right) \times 100$$

where:

B^C: The number of correctly located mappable balls.

B^W: The number of incorrectly positioned balls.

B^M: The total number of mappable balls.

L^C: The number of grid squares which do not contain a mappable ball that are correctly identified.

L^W: The number of grid squares which do not contain a mappable ball that are incorrectly identified.

L^T: The total number of grid squares which do not contain mappable balls.

b^V: The total value of the beams fired to solve the puzzle.

b^T: The total number of entry/exit positions of the Blackbox.

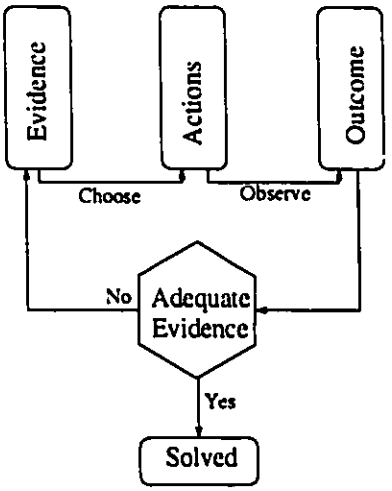


Figure 3.3: Diagnosis Type Problems

The *SCORE* metric assigns a numerical value to a result produced for any test case of Blackbox. A result with a lower score is considered to be of better quality than a result with a higher score. The weights placed on each factor rank the number of balls correctly identified as the most important factor, followed by the value of the beams fired, and the number of correctly identified grid squares (those not containing balls) is the least important.

Blackbox is an example of a diagnosis type problem (see Figure 3.3). Diagnosis type problems are structured such that there is a set of facts, and a set of actions available to the problem solver. The problem solver must examine the set of facts (or evidence) currently available, and based upon the evidence, the problem solver chooses one of the actions; then, the problem solver performs that action. The result of performing the action will be the introduction of new facts, which the problem solver can use to solve the problem. The problem solver can repeat this process until it is determine that there is adequate evidence available to reach a conclusion for the problem, or that none of the actions is likely to allow a conclusion to be reached.

Blackbox can be used to model many diagnosis type problems; for example the Blackbox puzzle could be used to model medical diagnosis: the beams fired by the problem solver are analogous to symptoms reported and diagnostic tests ordered by a physician, placing a ball is analogous to selecting a disease, and marking a square as empty is analogous to ruling out the possibility of a disease [30]. Our metric for the

quality of the result produced by the problem solver (*SCORE*) can be adapted to reflect the values that are appropriate in evaluating a problem solver for the problem that is being modeled, by adjusting the weights used in the *SCORE* metric. For example in the case of medical diagnoses, a physician might choose the weights used in the *SCORE* metric to make the selecting of the correct disease most important, followed by minimizing the number of tests ordered; this reflects the value that making a correct diagnosis is considered most important by physicians, and physicians are also sensitive to the fact that they should subject their patients to as few tests as possible. On the contrary, the administrators of private hospitals might not be that concerned about minimizing the number of tests ordered by physicians, and they may choose different weights for the *SCORE* metric than physicians.

Blackbox is an ill-structured problem. At first glance, representing the state of the Blackbox puzzle seems trivial. However, if we consider the set of hypotheses that must be constructed by the puzzle solver as part of the state of the problem, then representing the state of the Blackbox puzzle is not trivial [68]. When the puzzle solver selects a beam to be fired, the outcome of firing the beam is not known. Also, the number of states in the state space for the Blackbox puzzle is large because of the number of different ways in which the balls can be placed in the grid.

Blackbox is a problem that is suitable to be decomposed and solved by several puzzle solvers at the same time. The Blackbox puzzle can be decomposed geographically, and partial results to the Blackbox puzzle are possible as well as acceptable. A distributed version of the Blackbox puzzle has been created in which the Blackbox grid is divided into several regions, and a different problem solver is given the responsibility of determining the contents of the grid squares in each region [68]. Each problem solver has access to the grid squares in its region, and these grid squares constitute its local view. De Koven et al. have conducted several experiments in which teams of human problem solvers have cooperated to solve the distributed version of the Blackbox puzzle [69].

The Blackbox puzzle is a good choice as a sample problem for our testbed. The Blackbox puzzle meets our criteria for ill-structuredness, it can be decomposed and solved by several cooperating problem solvers, and we have a metric that can be

used to evaluate the result produced by a problem solver. In addition, Blackbox is cost effective for use in laboratory experiments: knowledge acquisition is not "too expensive", the time required by a human puzzle solver to solve a single Blackbox puzzle is not "too" long, and the results we obtain by studying the solving of the Blackbox puzzle are generalizable to other diagnostic type problems. The process of knowledge acquisition, required to construct a rule base for solving the Blackbox puzzle, is not too expensive because it is simple for a human to learn to solve Blackbox puzzles; thus, we are not obligated to consume the time of highly priced human experts in constructing the rule base. The time required for a person to solve a single test puzzle varies between 10 and 30 minutes; thus, the cost of collecting data concerning the ability of human puzzle solvers to solve a set of Blackbox puzzles is not prohibitive.

3.2 The Blackbox Expert

The Blackbox Expert is our rule-based system for conducting experiments in which we change the data items available to a single rule-based system, while we observe the functional and computational performance of the system [70]. The Blackbox Expert allows a human experimenter to specify a window (w_i) for the Blackbox Expert, changing the data items available as if the Blackbox Expert were a member of a CDPS, simulating the environment the rule-based system would face as a member of a CDPS. In addition, we expect to be able to reuse the components of the Blackbox Expert when we construct a test bed consisting of several cooperating rule-based systems, to be used in subsequent stages of our research.

In the types of experiments that we expect to conduct with the Blackbox Expert, an experimenter may wish to have the Blackbox Expert solve a single test case of the Blackbox puzzle, while the experimenter monitors the progress of the Blackbox Expert as it solves the puzzle. In other more complex experiments, the experimenter may wish to have the Blackbox Expert solve a set of test cases repeatedly, and each time the same test case is solved, a different number of data items are available to the Blackbox Expert. The features that must be included in the Blackbox Expert in order to permit an experimenter to conduct these types of experiments are as follows:

- the ability for the experimenter to set the data items available to the Blackbox

Expert.

- the ability for the experimenter to record the result produced by the Blackbox Expert for each test puzzle that it solves.
- the ability for the experimenter to evaluate the functional performance of the Blackbox Expert. The Blackbox Expert must apply the *SCORE* metric to each result it produces.
- the ability for the experimenter to monitor the computational performance of the Blackbox Expert. When dealing with rule-based systems, computational performance can be measured by recording the number of basic *operations* performed — inferences (or rules fired), and accesses to working memory [71]. The Blackbox Expert must record the number of rules fired and the number of facts it accesses as it solves each test puzzle.
- the ability for the Blackbox Expert to solve a single test case of the Blackbox puzzle, or solve a set of Blackbox puzzles without requiring human intervention.

The Blackbox Expert is composed of four modules: the User Interface, the Test Puzzle, the Current Hypothesis, and the Puzzle Solver, as shown in Figure 3.4 [72]. The User Interface allows the experimenter to control the Blackbox Expert. The Test Puzzle maintains the configuration of the current test puzzle that the Blackbox Expert is required to solve. The Puzzle Solver is a rule-based system responsible for solving Blackbox Puzzles, and the Current Hypothesis module contains the current state of the result that the Puzzle Solver is constructing as well as the entry and exit points for the beams that have been fired.

The Test Puzzle is responsible for simulating the problem domain. It contains the rules describing the basic principles of the interactions that can occur between the beams and the balls inside the Blackbox. The Test Puzzle maintains the data structure that contains the location of the balls within the Blackbox grid for the current test puzzle. The Test Puzzle receives from the Puzzle Solver (X, Y) coordinates of the entry point for beams that are fired by the Puzzle Solver. The Test Puzzle determines the trajectory of beams, and returns the (X, Y) coordinates of the exit point for the beams.

The Current Hypothesis module contains the Puzzle Solver's current hypothesis about the contents of the Blackbox grid squares, the certainty of any conclusion that has been drawn for the grid squares, and a list of the entry and exit points of the beams that have been fired (referred to as shot records). The Puzzle Solver

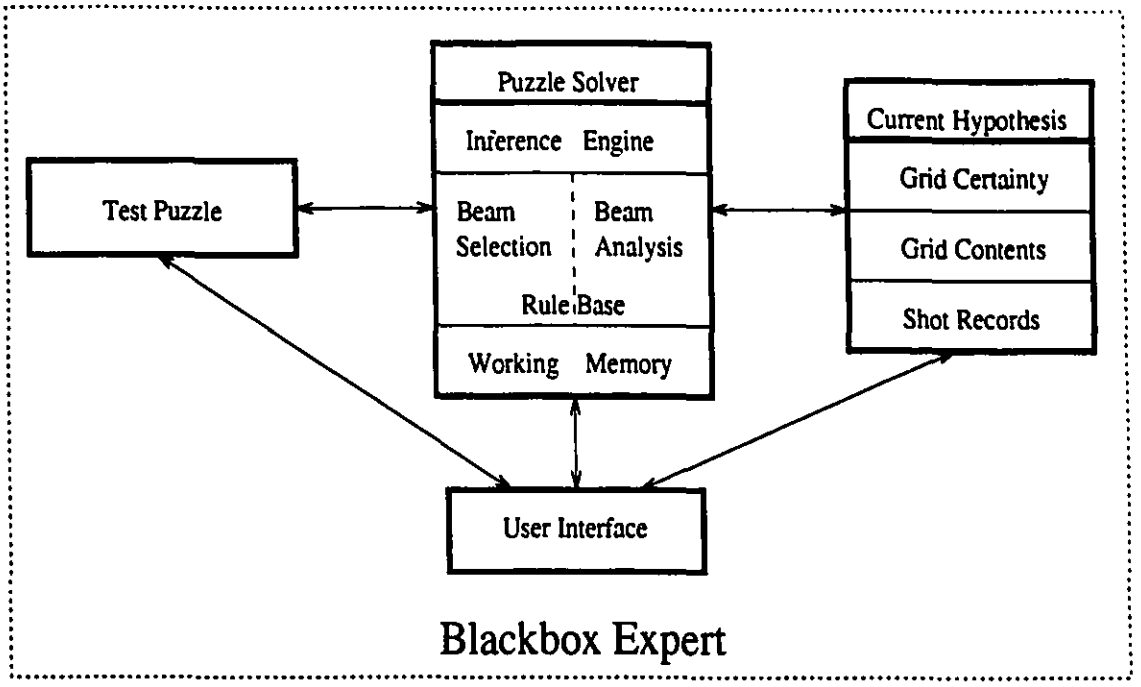


Figure 3.4: Structure of the Blackbox Expert

accesses the Current Hypothesis module only through a set of pre-defined functions, which specify the interface between the Current Hypothesis module and all the other components of the Blackbox Expert. These functions allow the Puzzle Solver to record a hypothesis for the contents of a grid square, to check the certainty of a hypothesis for the contents of a grid square, to check a region of the grid to see if it is known to be empty, etc.

The Current Hypothesis module provides the mechanism for controlling the availability of data items to the Puzzle Solver that we require for our experiments. The access functions defined for the Current Hypothesis module check the access privileges of the Puzzle Solver, before the Puzzle Solver is given access to any of the data items that are stored within the Current Hypothesis module. The access privileges of the Puzzle Solver are set by the human experimenter via the User Interface.

The Puzzle Solver is responsible for solving the Blackbox puzzle. The Puzzle Solver is composed of the standard modules found in rule-based systems: the rule base, an inference engine, and working memory. We make use of the CLIPS expert system shell in order to implement the Puzzle Solver [73]. CLIPS provides the inference engine required for the Puzzle Solver. CLIPS also manages the storage and

```
(defrule W91-24-left
  (phase selection)                                    ; Fire only during beam selection phase
  (poss-ball-found ? ?row1 ?col $?)
  (poss-ball-found ? ?row2&:(>= ?row2 (+ ?row1 2)) ?col $?)   ; The balls should be in the same
                                                       ; column, at least two rows apart
  (SHOTLEFT =(+ ?row1 1) 0 $?)                         ; A beam has not been fired that will
                                                       ; pass next to the upper ball
  (not (ADJUSTED-SHOT =(+ row?1 1) 0 W91-24-left)      ; Only adjust the beam once
  (test (isempty ?row1 1 ?row2 (-?col 1)))             ; Is the grid empty between the balls
                                                       ; and the edge of the box
=>
  (assert (ADJUST-SHOT =(+ ?row1 1) 0 50 W91-24-left 0))   ; Then adjust the value of the beam
)
```

Figure 3.5: Sample Beam Selection Rule

retrieval of facts in working memory, and the rule base is constructed using the rule language provided by CLIPS.

The Puzzle Solver uses a two phase approach when solving the Blackbox puzzle; the rule base of the Puzzle Solver is divided into two portions: beam selection, and beam analysis. The rules for beam selection encode the knowledge required to determine the next beam that is to be fired by the Problem Solver. The rules for beam analysis encode the knowledge required to analyze the beam entry and exit points, providing hypotheses about the contents of the grid squares. Beam selection corresponds to the planning phase (PP_i) of our problem-solving model for ill-structured problems, and beam analysis corresponds to the execution phase (E_i).

Figure 3.5 shows a sample beam selection rule, called `W91-24-left`. This rule will fire when there is a beam that has not been fired which would potentially pass between two balls, and the area of the grid that the beam would pass through is known to be empty. Once the rule fires, it asserts a fact that will cause an adjustment to the beam's rating. The pre-defined function `is_empty` is used by `W91-24-left` to access the Current Hypothesis module, and `is_empty` determines if the area of the grid where the beam would pass before it reaches the ball is empty.

The User Interface, shown in Figure 3.6, is responsible for handling the interaction between human experimenters and the Blackbox Expert. It allows the experimenter to monitor and assert control over the Blackbox Expert's progress as it solves the Blackbox puzzle. The User Interface consists of four areas: the Real Grid, the Hy-

pothesis Grid, the Dialog Window, and the Command buttons. The Real Grid allows an experimenter to view the contents of the Blackbox grid, as is contained in the Test Puzzle module. The Hypothesis Grid allows an experimenter to view the current hypothesis of the Puzzle Solver, as is contained in the Current Hypothesis module. The Dialog Window allows the Blackbox Expert to annotate the actions it takes to solve the puzzle, providing a trace of its actions for the experimenter to monitor. The Command buttons allow the experimenter to issue commands to the Blackbox Expert, setting the conditions for the different experiments that the experimenter wishes to conduct. The commands that can be issued by the experimenter are as follows:

- load a single test puzzle into the Test Puzzle module.
- solve the test puzzle that is currently stored in the Test Puzzle module, pausing after each shot is fired, until instructed to continue.
- solve the test puzzle that is currently stored in the Test Puzzle module, without pausing.
- set the window of the Puzzle Solver, to be enforced by the Current Hypothesis module, as the Puzzle Solver is solving test puzzles.
- save the result produced for a test puzzle, the *SCORE* of the result, and the computational performance of the Blackbox Expert when solving the test puzzle to a file.
- execute a sequence of commands which are contained in a file. This option is used by the experimenter to instruct the Blackbox Expert to solve a set of test puzzles.

When we construct a test bed consisting of several cooperating rule-based systems to be used in stage 2 of our research, we expect to be able to reuse the components of the Blackbox Expert, as shown in Figure 3.7. The cooperating rule-based systems will interact with each other when planning using the Decree and Consensus planning protocols, and share data using a blackboard. Each rule-based system has a window (w_i) on the blackboard. The Puzzle Solver module will be used to create the set of cooperating rule-based systems, and the working memory will contain the facts that will comprise the local view of the Puzzle Solver. The original rule base of the Puzzle Solver used in the Blackbox Expert will be decomposed to create the rule bases required for the set of rule-based systems, based upon the type of experiments that

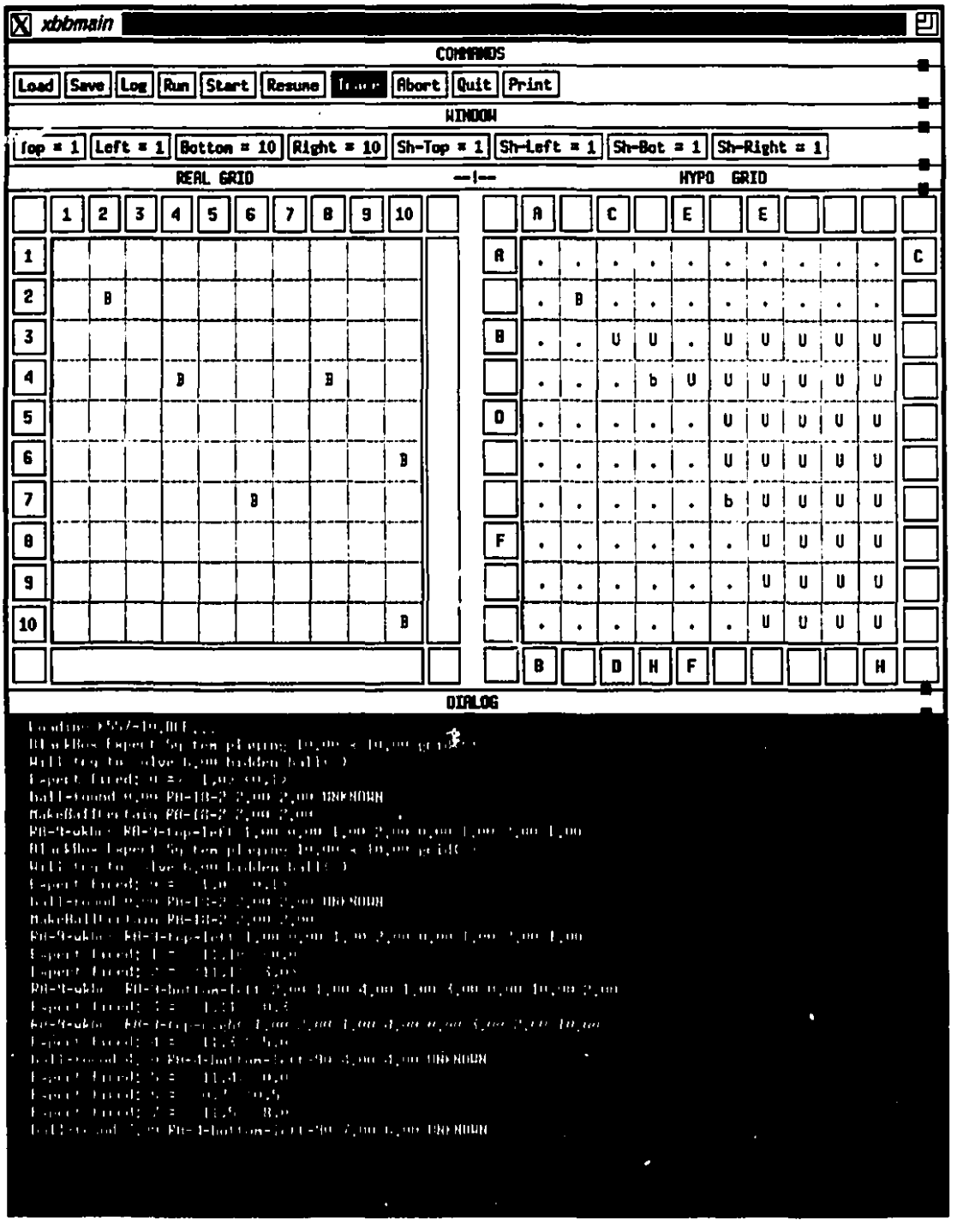


Figure 3.6: The Blackbox Expert's User Interface

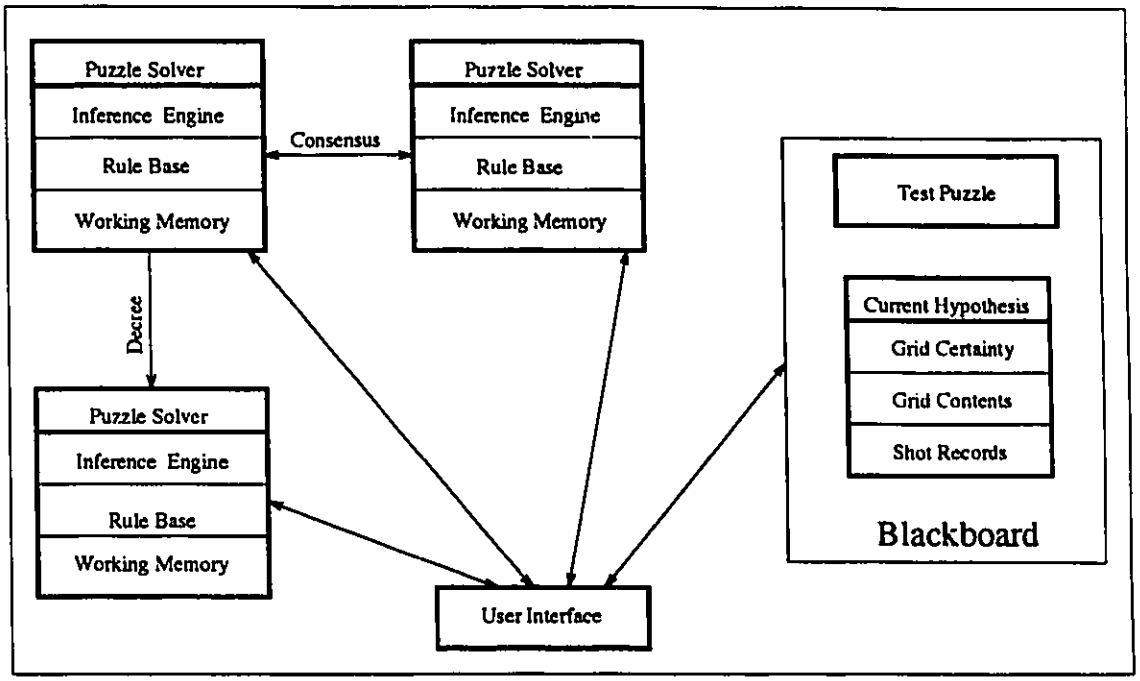


Figure 3.7: Testbed for Cooperating Rule-Based Systems

are to be carried out. The Current Hypothesis and the Test Puzzle module will be combined to form the blackboard used by the rule-based systems. The data structures contained in the Current Hypothesis module will be shared by all the rule-based systems, according to the access privileges that each rule-based system is granted, when the experimenter sets the window of each rule-based system. The beam exit points that are calculated by the Test Puzzle module will be placed on the blackboard, and the rule-based systems will only be able to access the shot records which are contained within their window. The User Interface will allow the experimenter to set a window for each rule-based system, and issue commands to the rule-based systems.

The Blackbox Expert permits experimenters to conduct experiments in which they can change the data items available to a single rule-based system, while observing the functional and computational performance of the system, in a cost effective manner. Our testbed permits an experimenter to observe the ability of the Blackbox Expert to solve a single Blackbox puzzle, or a set of puzzles, recording the result produced, recording the functional performance of the Blackbox Expert, and recording the computational performance of the Blackbox Expert. The Blackbox Expert is able to solve a set of test puzzles without requiring the intervention of the experimenter, and the

time required by the Blackbox Expert to solve a single test puzzle is between 20 and 30 seconds; thus, the cost of conducting experiments with the Blackbox Expert is not prohibitive. The development time for the Blackbox Expert is estimated to be about one person-year, which is "reasonable" given the resources available in our environment.

3.3 Validating the Performance of the Blackbox Expert

The validation of computer software systems refers to the process of determining if the system satisfies the need for which it was designed [74]. Validation of software systems is concerned with the factors that determine the system's usefulness, such as correctness of the results the system produces, its speed, efficiency, cost effectiveness, and many human factors. For our purposes, we will concentrate on validating only the functional performance of the Blackbox Expert.

The functional performance of the Blackbox Expert is validated by comparing the result it produces with the result produced by humans, for a specially designed set of test puzzles. A group of people, whose familiarity with Blackbox ranges from a few hours of exposure to several years of exposure, solved a set of test cases for Blackbox. Using the *SCORE* metric, each person received a rating for their ability to solve Blackbox puzzles. The Blackbox Expert solved the same set of test cases and was assigned a rating for its ability. Then, we compared the rating obtained by the Blackbox Expert with the ratings of the people, establishing the skill level of the Blackbox Expert.

The Blackbox puzzle includes several features that facilitate the validation of the Blackbox Expert. Any result that is proposed by a person for a test case of the Blackbox puzzle can be evaluated using the *SCORE* metric. The time consumed by humans to solve each test case of Blackbox is between 10 minutes for someone with a lot of experience, and 30 minutes for a beginner. The time required for the Blackbox Expert to solve a test puzzle varied between 20 and 30 seconds. Therefore, validation of the Blackbox Expert is not costly. Developing a population of human experts against whose performance the Blackbox Expert can be validated is simple

because the effort required by a human to become skilled at solving Blackbox is not prohibitive.

Kristina Pitula and Carol De Koven (both having several years of experience in solving Blackbox puzzles) designed the set of test cases used for validating the Blackbox Expert. The puzzles in the test set were given a rating of easy, medium, or hard. Kristina Pitula and Carol De Koven also participated in a group discussion with several other people who also had a lot of experience with Blackbox. The group focused its discussion on the factors that would determine the degree of difficulty of a test case of the Blackbox puzzle. Using the input from this discussion, the two people responsible for the test set determined the criteria used to develop the test set, and placed each test case that was developed into one of the three categories (easy, medium, and hard).

Kristina Pitula and Carol De Koven decided that the following features would contribute to the complexity of a test case:

- the presence of unmappable grid squares.
- beam entry and exit points that can be accounted for by many different trajectories through the grid.
- the presence of balls in the corners of the box.
- a positioning of balls that results in a large number of hits and reflections.

The presence of unmappable grid squares increases the complexity of a test case because it makes it difficult to decide when a result has been found. Many people seem to have an aversion to leaving parts of the Blackbox grid unknown. They will actually convince themselves that they are able to pinpoint the locations of balls which actually are unmappable. People tend to always choose the simplest solution; thus, when there are many possible trajectories that can account for the entry and exit points of a beam, people tend to make errors. If people do not confirm their hypotheses for the locations of the balls by firing more beams, they risk making mistakes. The strategy used by many people when solving Blackbox is to work from the edges of the grid towards the center. Balls in the corners of the grid prevent a person from following this strategy, thereby increasing the difficulty of solving the puzzle. Deflections provide a lot of information than hits or reflections because deflections often pinpoint the location of

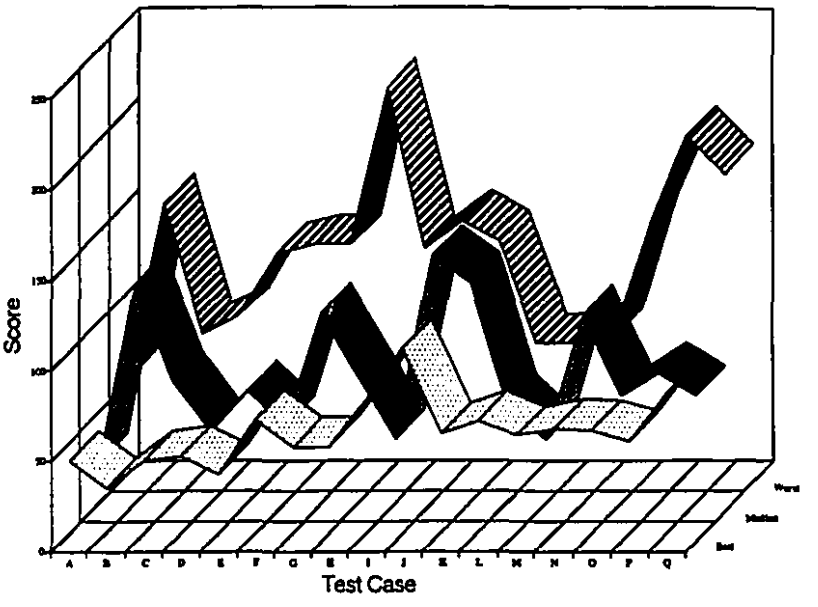


Figure 3.8: Best, Median, and Worst Player

a ball, and often indicate that many grid squares are empty. A puzzle in which the positioning of the balls results in many hits and reflections is very difficult to solve, as there is very little information with which one can determine the contents of the grid.

The rating of each person who participated in our validation experiment was done in two stages: a *learning* stage, and a *puzzle solving* stage. The learning stage included a set of instructions explaining the basic principles of the Blackbox puzzle as well as the *SCORE* metric, a demonstration of how a person would solve the puzzle, and a set of sample puzzles designed to demonstrate the principles and the *SCORE* metric. The puzzle solving phase required each person to solve the test cases, which were presented to them in a random order. Even the people with a lot of previous exposure to Blackbox were required to go through the learning phase, in order to ensure that they fully understood the *SCORE* metric.

Fifteen people participated in our validation experiment. They solved the 17 test cases in our Blackbox test set. Figure 3.8 shows the scores obtained by the people who had the best (lowest), median (middle), and worst (highest) average score for all seventeen test cases. Test cases A, B, and C were placed in the easy category, cases D,

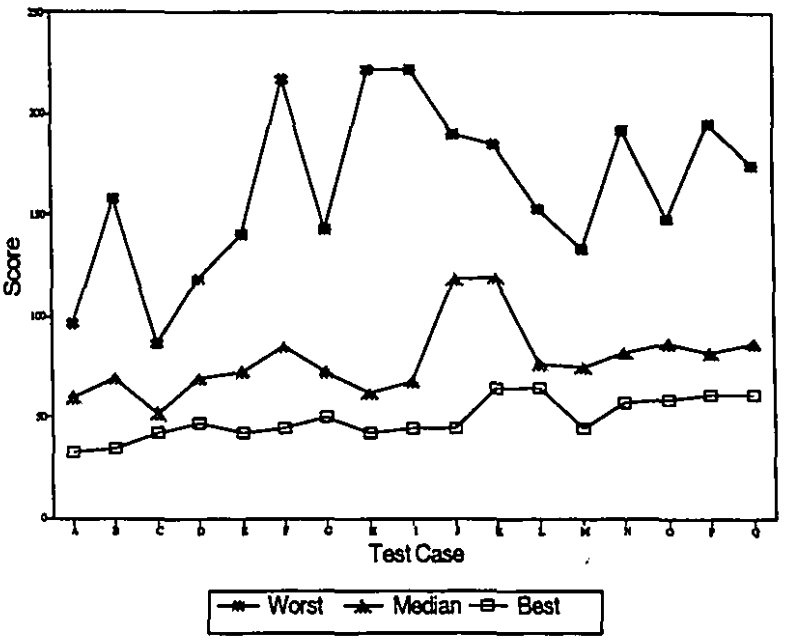


Figure 3.9: Best, Median, and Worst Scores

E, F, and G were placed in the medium category, and the other test cases were placed in the hard category. The person with the best score was also found to be the most consistent puzzle solver; this consistency is evident from the narrow fluctuation in the scores for the person. The scores for the best person tend to rise slowly from easy to hard test cases. The person at the median has scores that fluctuate more widely than the best player. The person with the worst average score also experiences the largest variation in score.

The best, worst, and median scores for each test case are shown in Figure 3.9. Again, the lowest scores obtained by any person exhibit the least variation. The median scores vary more than the best scores and the worst scores have the largest variation. These scores also exhibit an upward trend when the easy, medium, and hard test cases are compared.

The average scores and total number of errors made in placing balls in the Black-box grid by the people who solved the test set are shown in Table 3.1. Both the average score and total errors made in placing the balls increase when comparing the easy, medium, and hard test cases. As expected, this trend seems to suggest that the

	Total Ball Errors			
SUBJECT	EASY	MED	HARD	ALL
1	8.0	4.0	18.0	30.0
2	0.0	0.0	2.0	2.0
3	0.0	0.0	9.0	9.0
4	0.0	0.0	4.0	4.0
5	2.0	1.0	9.0	12.0
6	2.0	2.0	23.0	27.0
7	0.0	0.0	6.0	6.0
8	0.0	6.0	8.0	14.0
9	0.0	0.0	7.0	7.0
10	0.0	6.0	21.0	27.0
11	0.0	8.0	17.0	25.0
12	0.0	4.0	13.0	17.0
13	2.0	20.0	33.0	55.0
14	8.0	15.0	30.0	53.0
15	10.0	22.0	27.0	59.0
AVG	2.1	5.9	15.1	23.1

	AVG SCORE			
SUBJECT	EASY	MED	HARD	ALL
1	80.0	72.2	84.7	80.9
2	60.0	66.3	69.3	66.9
3	45.0	56.3	73.5	64.4
4	69.2	74.4	79.1	76.2
5	64.7	80.1	82.7	78.9
6	50.0	65.2	97.8	81.7
7	57.5	62.5	78.2	70.8
8	61.7	107.0	88.1	87.9
9	59.2	62.5	78.6	71.4
10	59.2	80.8	93.3	84.3
11	66.7	93.9	114.7	101.4
12	54.2	76.6	93.5	82.6
13	46.7	137.4	130.4	117.3
14	86.1	118.2	124.4	116.2
15	102.6	125.2	141.1	130.6
AVG	64.2	85.2	95.3	87.4

Table 3.1: Average Score and Total Errors in Placing Balls

performance of the people when solving the test cases from each of the three groups in our test set is different. In order to validate this assumption, we performed an analysis of variance to determine if the difference that is observed in the mean scores of the three groups can be accounted for by the variance in the scores of all the test cases solved. The ANOVA table is shown in Table 3.2. The F ratio obtained with 2 and 252 degrees of freedom is 13.42. An F ratio of 4.69 or greater is needed for significance with a confidence level of 99%, thus we can reject the null hypothesis that $\mu_{easy} = \mu_{medium} = \mu_{hard}$.

Having determined that the average scores for the three groups are statistically different, we must now examine the individual differences between the groups. Table 3.2 shows the confidence intervals for the pairwise comparisons of the means of the groups in the test set. These comparisons are done using an F value of $F_{2,252,.95} = 3.035$. As the confidence intervals for $\mu_{easy} - \mu_{medium}$ and $\mu_{easy} - \mu_{hard}$ do not contain zero we can reject the null hypotheses $(\mu_{easy} - \mu_{medium}) = 0$ and $(\mu_{easy} - \mu_{hard}) = 0$. Thus, $\mu_{easy} < \mu_{medium}$ and $\mu_{easy} < \mu_{hard}$. However, the confidence interval for $\mu_{medium} - \mu_{hard}$ does contain zero, which does not permit us to reject the null hypothesis $(\mu_{medium} - \mu_{hard}) = 0$. Thus, there is a statistical difference between the easy and medium groups, the easy and hard groups, but not the

Source of Variation	Sum of Squares	df	Mean Square	F
Between group	33915.97	2.00	16957.99	13.42
Within group	318506.48	252.00	1263.91	
Total	352422.45	254.00		

	Confidence	Interval
Pairwise Comparison	Lower Limit	Upper Limit
Easy-Medium	-38.34	-3.79
Medium-Hard	-23.44	3.32
Easy-Hard	-46.01	-16.24

	Confidence	Interval
Group Comparison	Lower Limit	Upper Limit
Easy-(Medium,Hard)	-40.77	-11.42
Hard-(Easy,Medium)	9.38	31.81

Table 3.2: Analyses of Variance

medium and hard groups. The factors we used to place the different test cases into the groups are valid. However, the difference between the medium and hard groups is not confirmed.

The last set of tests to be performed on our test set are the group comparisons shown in Table 3.2. The confidence intervals are given for $\mu_{easy} - (\frac{\mu_{medium}+\mu_{hard}}{2})$ and $\mu_{hard} - (\frac{\mu_{easy}+\mu_{medium}}{2})$ using an F value of $F_{2,252,.95} = 3.035$. In both cases, we can reject the null hypothesis that $(\mu_{easy} - (\frac{\mu_{medium}+\mu_{hard}}{2})) = 0$ and $(\mu_{hard} - (\frac{\mu_{easy}+\mu_{medium}}{2})) = 0$. Thus, the easy group is different from the average of the medium and hard groups and the hard group is different from the average of the easy and medium groups.

The performance of the Blackbox Expert on the test cases is shown in Figure 3.10. The Blackbox Expert is compared to the people who were rated as the best, median, and worst players. Except for one test case (I), the best player performed better than the Blackbox Expert. The Blackbox Expert performed better than the worst player in 15 of the 17 test cases. The Blackbox Expert performed better than the median player in 7 of the 17 test cases. The two test cases (C and P) where the Blackbox

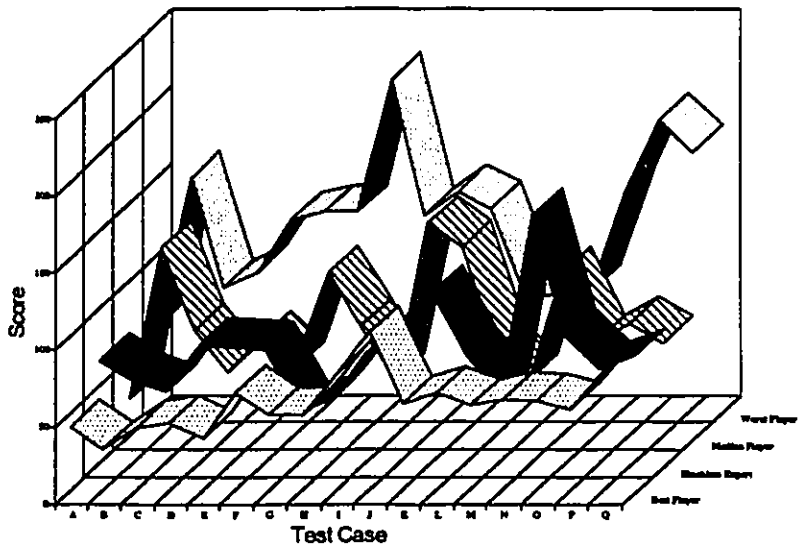


Figure 3.10: Scores of the Blackbox Expert

Expert performed poorly compared to the worst player indicate a deficiency in the knowledge base. Both test cases C and P have balls located near the corners of the Blackbox grid. The knowledge base of the Blackbox Expert is lacking rules to determine when balls are located near the corners of the Blackbox.

The average score and the total number of errors made placing balls by the humans and the Blackbox Expert are shown in Table 3.3. The Blackbox Expert on average made fewer errors locating balls than the humans. The lowest total number of errors

	Average Score			
	EASY	MEDIUM	HARD	ALL
PEOPLE	64.2	85.2	95.3	87.4
EXPERT	64.2	73.6	88.9	80.9
	Average Ball Error			
PEOPLE	2.1	5.9	15.1	23.1
EXPERT	0.0	2.0	12.0	14.0

Table 3.3: Average Score and Ball Errors

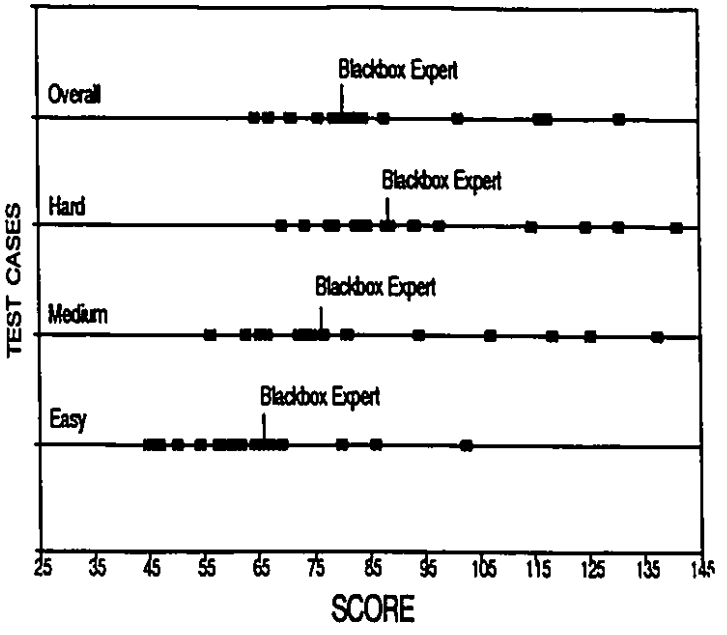


Figure 3.11: Blackbox Expert vs Humans

(2 errors in 17 test cases) was made by a person with several years of experience solving the Blackbox puzzle as seen in Table 3.1. Also, the Blackbox Expert had a better average score on each group of test cases in the test set except on the easy test cases where it had the same average score. When the total number of errors in locating balls is considered, the Blackbox Expert ranks 7^{th} compared to the 15 people.

The average score obtained by the Blackbox Expert for each group of test cases as well as the entire test set is compared to the average score obtained by the people in Figure 3.11. The Blackbox Expert ranks 10^{th} on the easy test cases, 7^{th} on the medium and hard test cases, and 7^{th} on the entire test set. The improvement observed in the Blackbox Expert's ranking on the medium and hard test cases occurs because even though the Blackbox Expert and the humans can find all the balls in the easy test cases, the Blackbox Expert requires more beams to solve the test cases. In the case of the medium and hard test cases, the Blackbox Expert still tends to fire more beams than the humans. However, on average it makes fewer errors, which allows it to improve its position.

Considering the types of experiments that we intend to conduct using the Black-

box Expert, the results of our validation experiment indicate that the functional performance of the Blackbox Expert is acceptable. The Blackbox Expert's ability to solve Blackbox puzzles is superior to the ability of many humans, but not superior to humans with a lot of experience solving Blackbox puzzles. We believe that the ability of the Blackbox Expert is adequate, permitting us to to study the Blackbox Expert's ability to solve Blackbox puzzles, as we vary the data items available. We believe that the Blackbox Expert can exhibit the reduction in performance that we expect when the number of data items available to the system is reduced, because its ability to solve test puzzles is respectable when compared to the ability of humans to solve the same test puzzles.

3.4 Data Distribution and Performance: An Experiment

Given the existence of the Blackbox Expert, we can now conduct experiments specifically to examine the effects of data distribution on performance. Based upon our survey of the literature, in chapter 2 we discussed the trends in the change in performance that were expected by researchers as the data distribution chosen for a CDPS is varied by the CDPS designer. However, we lacked evidence that the expected trends will actually occur in operational rule-based systems. In this section, we present results from an experiment that quantifies the effects of data distribution on the performance of the Blackbox Expert [31]. We then discuss how the insight gained from our experiment may be used by a CDPS designer, when selecting the data distribution in a CDPS for solving a diagnosis type problem.

3.4.1 Experimental Design

We conducted our experiment using the Blackbox Expert to solve a set of randomly selected test cases of the Blackbox puzzle, and measured the Blackbox Expert's performance. We used randomly selected test cases in order to avoid introducing any bias into the experiment. While our experiment is conducted using the Blackbox Expert to solve a given set of test puzzles, we also determine the implication of the results from this experiment on the entire population of Blackbox puzzles; we statistically analyze

the data obtained in this experiment to discover if any of the trends observed in the data collected using the randomly selected test puzzles are statistically significant. A statistically significant trend indicates that the trend observed in the data obtained from our set of test puzzles is representative of the performance we can expect when the Blackbox Expert solves any Blackbox puzzle.

Our experiment involved having the Blackbox Expert solve 20 randomly selected puzzles. Each puzzle is solved 10 times with a reduced number of data items available (measured using the information deficit metric) on each successive run. The Blackbox Expert started by solving each puzzle with an information deficit of zero (all data items are available). On each successive run, the information deficit faced by the Blackbox Expert was increased by 0.1. Each time a test puzzle is solved by the Blackbox Expert, we recorded the Blackbox Expert's functional and computational performance.

The hypotheses that we tested in our experiment considered the functional and computational performance of the Blackbox Expert in both problem-solving phases: beam selection (PP_i), and beam analysis (E_i). The effects of an **increase in information deficit** on **computational performance** are hypothesized to be as follows:

H_1: the processing resources consumed by a rule-based system to process the data items required to solve a subproblem decreases. In the case of the Blackbox Expert, the processing resources consumed to solve a subproblem is given by the average number of rules fired during each beam analysis phase, and is denoted by E_R.

H_2: the communication resources consumed by a rule-based system to process the data items required to solve the subproblem decreases. In the case of the Blackbox Expert, the communication resources consumed to solve a subproblem is given by the average number of accesses made to the Current Hypothesis module during beam analysis, and is denoted by E_A.

H_3: the processing resources consumed by a rule-based system to create a plan decreases. In the case of the Blackbox Expert, the processing resources consumed to create a plan is given by the average number of rules fired during beam selection, and is denoted by P_R.

H_4: the communication resources consumed by a rule-based system to create a plan decreases. In the case of the Blackbox Expert, the communication resources consumed to create a plan is given by the average number of accesses to the Current Hypothesis during beam selection, and is denoted by P_A.

The effects of an **increase in information deficit** on **functional performance** are hypothesized to be as follows:

H_5: the number of errors in the results produced by the rule-based system for the subproblem increases. In the case of the Blackbox Expert, the number of errors in the results produced is given by the number of *ball errors*, and is denoted by B_E. A ball error occurs whenever a ball in the Blackbox grid is not located by the Blackbox Expert, or the Blackbox Expert concludes that a grid square contains a ball, when in fact it does not.

H_6: the rate of plan failures increases. In the case of the Blackbox Expert, a plan failure occurs whenever a beam is fired and then no changes are made to the grid during the next beam analysis phase. The plan failure rate is $\left(\frac{\text{plan failures}}{\text{beam selection phases}} \times 100\right)$, and is denoted by P_F.

H_7: the quality of the results produced by the rule-based system decreases. In the case of the Blackbox Expert, the quality of the results produced is given by the *SCORE* metric.

We statistically analyzed the data obtained (curves for functional and computational performance) to determine its use in establishing the performance that we can expect with the Blackbox Expert solving any Blackbox puzzle; using a multivariate repeated measures design, we performed an analysis of variance on the data collected for each measure [75]. A separate analysis was performed for each measure, because the hypotheses that we tested in our experiment do not consider any joint effects of information deficits. The contrasts that we chose to use for the analysis of variance are as follows:

- the measures for computational performance (E_A, E_R, P_A, and P_R) were tested using the standard polynomial contrasts. The polynomial contrasts indicate if there exists a statistically significant trend in the data being tested. A statistically significant trend identified in the data would allow us to establish the trends that could be expected in the Blackbox Expert's computational performance as the information deficit changes, when the Blackbox Expert solves any Blackbox puzzle.
- The measures for functional performance (SCORE and B_E) were tested using repeated pairwise contrasts. The repeated pairwise contrasts test for a statistically significant difference in the data being tested. A statistically significant difference identified in the Blackbox Expert's functional performance solving the given set of test puzzles would allow us to establish the changes in the Blackbox Expert's functional performance that could be expected, when the Blackbox Expert solves any Blackbox puzzle as the information deficit changes.

- the measure for the rate at which plans fail (P_F) was tested using the Helmert contrasts. The Helmert contrasts test for a significant difference between the plan failure rate observed at each information deficit level and the average plan failure rate that occurred with larger information deficits. A statistically significant difference identified in the Blackbox Expert's plan failure rate would allow us to establish the changes in the Blackbox Expert's plan failure rate that could be expected, when the Blackbox Expert solves any Blackbox puzzle as the information deficit changes.

3.4.2 Experimental Results

The trends that were expected for the computational performance of the Blackbox Expert are evident in the raw data. The means for the measures of computational performance during beam analysis (E_A and E_R) obtained from the 20 puzzles solved at each information deficit are shown in Figure 3.12. The number of accesses to the Current Hypothesis module and number of rules fired during beam analysis decrease as the information deficit increases. The means for the measures of computational performance during beam selection (P_A and P_R) obtained from the 20 puzzles solved at each information deficit are shown in Figure 3.13. The number of accesses to the Current Hypothesis module decreased as the information deficit increased, and the number of rules fired during beam selection decreased as the information deficit increased, which is consistent with the fact that the Blackbox Expert uses a planner that operates by rating a set of goals.

The trends that were expected for the functional performance of the Blackbox Expert are evident in the raw data. The means for the measures of functional performance of the Blackbox Expert ($SCORE$, B_E, and P_F) obtained from the 20 puzzles solved at each information deficit are shown in Figure 3.13 and Figure 3.14. The quality of the result produced by the Blackbox Expert is reduced as the information deficit increases (Figure 3.14), and the number of ball errors also increased with an increase in information deficit (Figure 3.14). We also observed an increase in the plan failure rate as the information deficit increased (Figure 3.13).

Significant trends in the Blackbox Expert's computational performance were found (see Appendix A), with $\alpha = 0.05$. The strongest significant trends found for each measure are as follows: in P_A, linear and quadratic; in P_R, linear and cubic; in E_A,

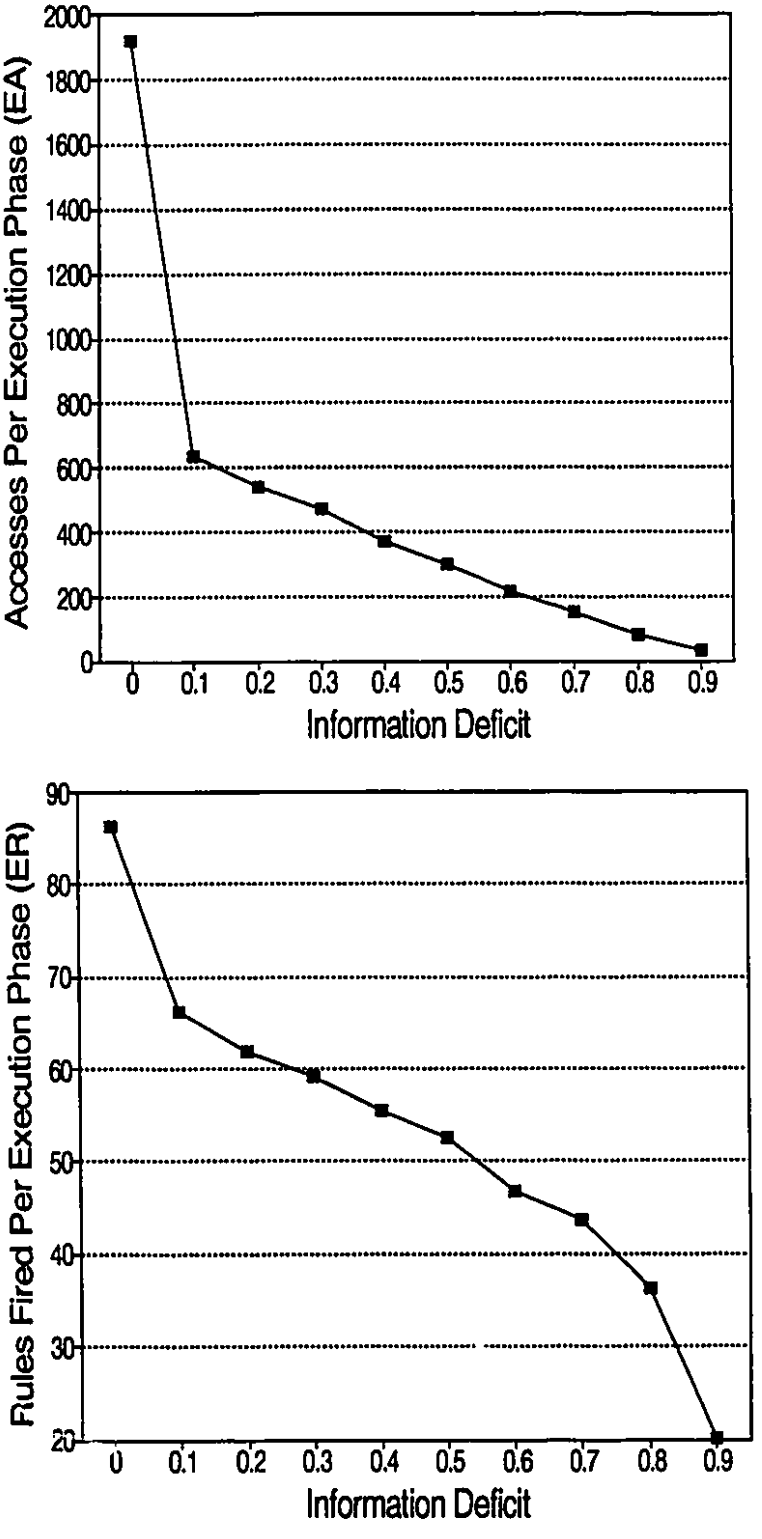


Figure 3.12: Computational Performance – Execution

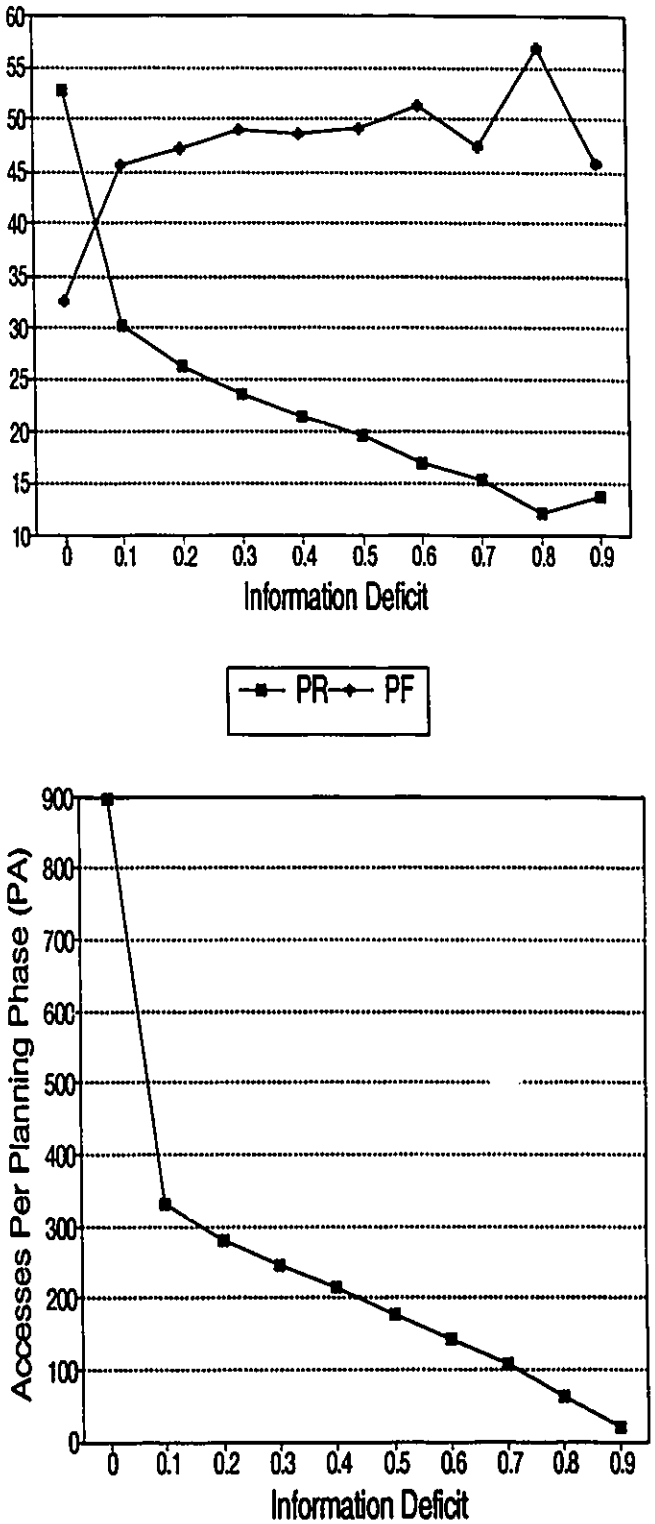


Figure 3.13: Performance - Planning

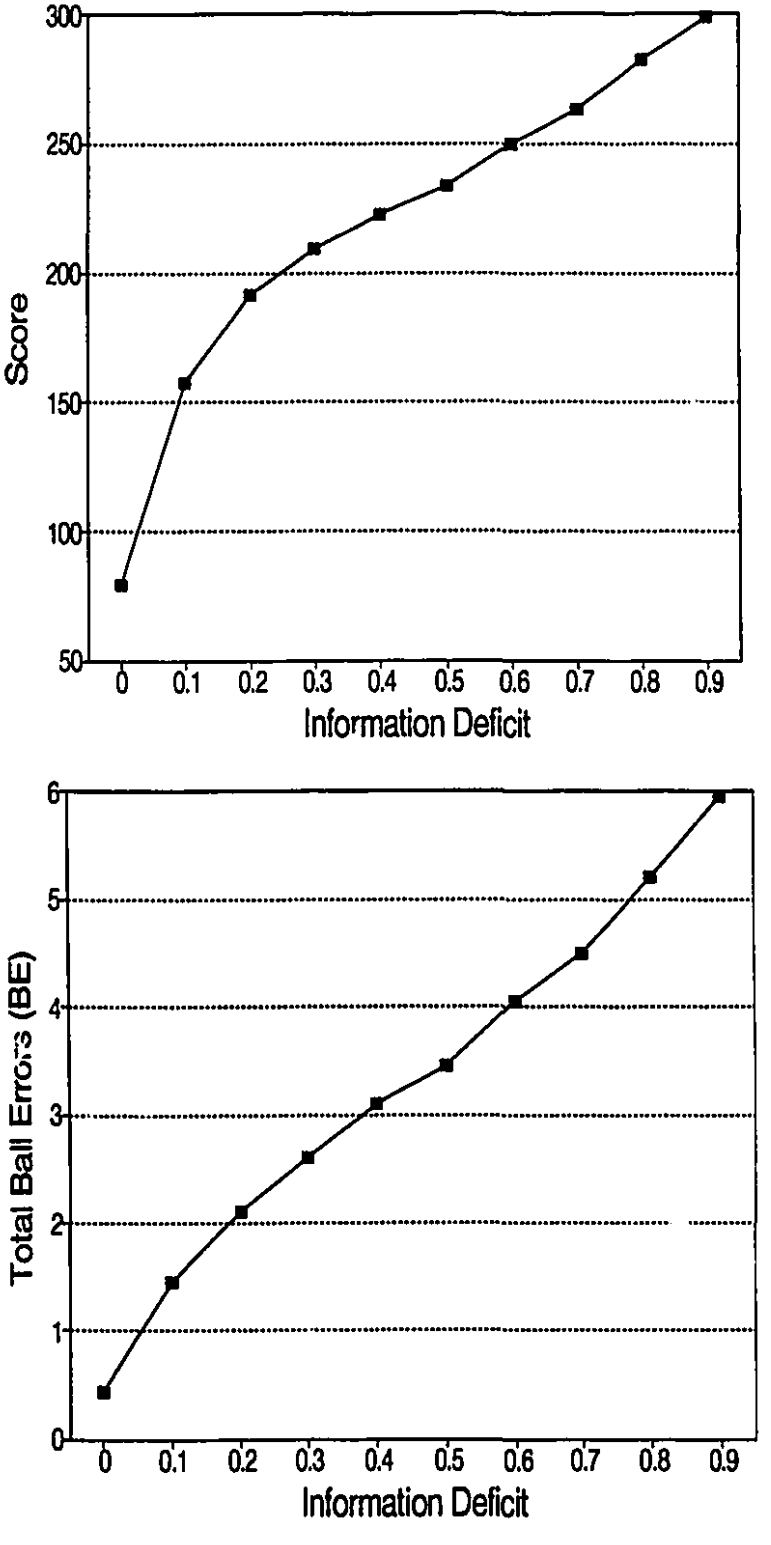


Figure 3.14: Functional Performance

linear and quadratic; and in E_R, linear and quadratic. The trends for the measures P_A, E_A, and E_R indicate that we can expect the Blackbox Expert's performance on these measures when solving any Blackbox puzzle to follow a curve with a shape that is composed of linear and quadratic polynomials, and the curve will decrease as the information deficit increases. The trends for the measure P_R indicates that we can expect the Blackbox Expert's performance on this measure when solving any Blackbox puzzle to follow a curve with a shape that is composed of linear and cubic polynomials, and the curve will decrease as the information deficit increases.

Significant differences in the Blackbox Expert's functional performance were found in measures $SCORE$, B_E, and P_F (see Appendix A), with $\alpha = 0.05$. $SCORE$ and B_E show statistically significant differences in the Blackbox Expert's functional performance at each information deficit used in the experiment. The trends for the measures $SCORE$ and B_E indicate that we can expect a change in the Blackbox Expert's functional performance when solving any Blackbox puzzle, if the information deficit changes by at least 0.1. Significant differences in the Blackbox Expert's plan failure rate P_F occur only when the information deficit is very small or very large; thus, we only expect a change in the Blackbox Expert's performance on this measure when solving any Blackbox puzzle if the information deficit is small or large, and a change in the information deficit occurs.

3.4.3 Discussion of Results

This experiment studied the effects of an information deficit on the computational and functional performance of the Blackbox Expert. Using a set of randomly selected Blackbox puzzles, the relationships between the information deficit faced by the Blackbox Expert and its performance have been established quantitatively. The analysis of variance tests used in the experiment showed that the effects of information deficits on the performance of the Blackbox Expert are significant at $\alpha = .05$, establishing the effects of an information deficit on the performance of the Blackbox Expert that can be expected when the Blackbox Expert solves any Blackbox puzzle.

The performance curves we measured in our experiment can aid the CDPS designer in selecting the data distribution in an organization imposed on a set of co-

operating rule-based systems. While our experiment was based on the Blackbox Expert solving Blackbox puzzles, we believe that these results are relevant to other rule-based systems solving diagnoses type problems. For example, the statistically significant linear and quadratic trends we measured in the number of accesses required by a rule-based system during execution (E_A) indicates that we can expert that there would be a large change in E_A when the information deficit experienced by the rule-based system varies between 0 and 0.1 as compared to the change in E_A when the information deficit varies between 0.1 and 0.8. We expect that the performance curves that were obtained for the Blackbox Expert to be similar to the performance curves of other rule-based systems that solve diagnoses type problems in the same manner as the Blackbox Expert; thus, the results of our experiment can be used as a model for the effects of data distribution on rule-based systems solving diagnoses type problems.

3.5 Conclusion

Testbeds are required for experimental work that are flexible and cost effective. The Blackbox Expert is a testbed that can support experiments in which we can change the data items available (a_i) to a single rule-based system, and observe the functional and computational performance of the system. The cost of constructing the Blackbox Expert as well as the costs of conducting experiments using the Blackbox Expert is not prohibitive. A human can become proficient at solving the Blackbox puzzle (the problem solved by the Blackbox Expert) in a few days. Therefore, we do not have to rely on human experts in constructing the rule base of the Blackbox Expert, or its validation. The time required for humans or the Blackbox Expert to solve the Blackbox puzzle is not too costly.

Using the Blackbox Expert, we conducted an experiment to examine the effects of data distribution on performance. Our experiment establishes that data distribution is an important factor determining the performance of the rule-based systems in a CDPS, when the agents in the CDPS are implemented as rule-based systems. Our experiment provides evidence that the trends expected by researchers for the change in performance as the data distribution in an organization chosen for a CDPS is varied by

the CDPS designer do actually occur in operational rule-based systems, and provides evidence that the effects of data distribution on performance are important for the Blackbox puzzle as well as diagnoses type problems. Confirming the importance of data distribution in determining performance represents a step forward; the CDPS designer must be very careful when selecting the data distribution in an CDPS.

The results of our experiment shed some light on the effects of data distribution on performance. Yet, the CDPS designer must be cautious in using the trends and relationships measured in our experiment for estimating the effect of data distribution on other rule-based systems. There does not exist any well known theory by which a CDPS designer can evaluate the similarities between different rule-based systems, and the problems they solve; thus, the CDPS designer must rely on intuition to decide if the effect of the data distribution we measured on the performance of the Blackbox Expert is indicative of the effect of data distribution on other rule-based systems.

Chapter 4

A Model for Rule-Based Systems

"If I have been further, it is by standing on the shoulders of giants."
- Sir Isaac Newton

Thus far in our study of CDPS systems, we have focused on an inter-agent perspective, which is concerned with issues relating to the manner in which the agents in a CDPS interact: the manner in which a problem is decomposed, the tasks assigned to each agent, the sharing of data items both during problem-solving, and the planning protocols used by the agents. We have introduced a model that permits the CDPS designer to specify the manner in which data items are shared by the rule-based systems in an organization during both phases of problem-solving: planning (CW) and execution (WS). Our organizational model specifies the data items that are available to a rule-based system within its window (w_i).

We have not considered the internal structure of rule-based systems that are to be agents in a CDPS (the intra-agent perspective); we believe that such a model is required to aid the CDPS designer in determining how to set the availability of specific data items in order to improve the performance of the rule-based systems. Our intent is to use this model to capture the data items required by the system to produce specific results (discussed further in chapter 5). An understanding of the data items required by a rule-based system to produce specific results will be useful to the designer of CDPS when specifying the availability of data items in the CDPS.

Researchers in the field of database design have modeled the structure of a rule base in order to address the problems of efficient compilation, storage, and access to

large sets of rules [76]. Their assumption is that the underlying mechanism for storage of rules is the relational schema. The goal of the research in that area is to determine the rules required to answer a particular query as well as the most efficient method for building a response to a query. Determining which rules are required to answer a particular query is necessary, given the assumption that not all rules are resident in the memory of the computer. The notion of efficiency that is used to determine the rule sequence that is best suited for answering a query is based upon assumptions about the relationships between the relational operators that are used to construct an answer for the query. Based upon these assumptions, researchers have proposed a notion of *access paths* to capture the dependencies that exist among the rules in the database. The models proposed by the researchers in this field are not adequate for our purposes.

The rule-based expert system validation literature reports a number of approaches for defining the structure of a rule base as chains of inter-dependent rules, called execution paths. The EVA system [77] defines a dependency graph (DG) that is used to generate test cases for validating a rule-based 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 identified in the DG which do not reflect sequences of rules that fire when a rule base is exercised. Rushby and Crow [78] propose a refinement to the DG used in EVA, where the rule-dependency relation is improved, but under certain conditions the improvement obtained is still unsatisfactory, and suffers from the same problem as the DG used in EVA. A stricter method for determining rule dependencies is proposed by Kiper [79], 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, which prohibits the use of this method on large rule bases.

We believe that the best approach for capturing the structure of a rule-base that can be used to identify the data items required by a rule-based system to produce specific results is to consider a rule-base to be composed of a set of chains of inter-dependent rules. Based upon our survey of previous attempt by researchers to define

the notion of a path in a rule base, we believe that a path must satisfy the following criteria:

Accuracy There are two aspects to accuracy in defining paths:

- 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.[1]
- each path must correspond to sequences of rule firings *that can actually occur at run-time.*

Meaningfulness When the rules forming a path fire, their combined actions should carry out a function as intended by the knowledge engineer, which is seen as having significantly advanced the state of the problem being solved.

Computational Tractability In order to enable efficient automatic generation of the paths in a rule base, we require that the computational effort involved in finding the rules that comprise a path be small, and the number of paths that will be generated for a rule base must be computable; that is, we want to prohibit a combinatorial explosion in finding paths.

In this chapter, we describe a formal model for the structure of a rule-based system, and this model meets the above three criteria [33, 34, 35]. Then we present the algorithms used to implement Path Hunter, our rule base analysis tool which identifies the set of paths contained in a given rule-base; we show that Path Hunter will identify all the paths that exist in a given rule base. We then discuss how the knowledge engineer can control combinatorial explosions when using Path Hunter to analyze a rule base. We then use Path Hunter to analyze the rule base of the Blackbox Expert, identifying all the paths that exist in its rule base.

4.1 Rule Base Structure

It is our belief that problem solving requires two entities: the problem to be solved, and the artifacts designed to solve the problem. Previous efforts by researchers characterizing the structure of a rule-based system focused solely on the artifacts designed to solve the problem, that is, the rule base. Our model considers both entities: the

[1] This may seem obvious, but not all previous proposals are sufficiently well-specified, as observed in [78].

problem to be solved, and the rule base designed to solve that problem. When modeling the structure of the rule-based system, we will make use of the interaction between the structure of the problem to be solved and the structure of a rule-base that is designed to solve that problem. The structure of the problem to be solved will introduce a semantics that defines a notion of meaningfulness and completeness for the structural components of the rule-based system captured by our model.

When constructing a rule-based system, the expertise possessed by the domain expert is embodied by the knowledge engineer into a set of rules. We consider a rule-based system $\mathcal{E}$ to be a triple $\langle E, RB, WM \rangle$ where: E is an inference engine, RB is the set of rules for solving P^I, and WM is the working memory where facts (representing current data) are stored. 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. The state of $\mathcal{E}$ is denoted by the set of facts present in WM. When a rule fires, it changes the state of the system by adding or removing facts from the WM.

The structure of the facts used by $\mathcal{E}$ is important in representing the state of the problem being solved. Facts consist of a predicate name and a list of arguments. Each fact is an instance of a factor of P^I (a_i). The predicate in a fact indicates the relationships that exist among the arguments of a fact as well as which factor of P^I that the fact is an instance of. 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: $f_i = \langle \lambda_i, l_i \rangle$ such that l_i is a list of data elements, and $\lambda_i \in R$ identifies the relationship between the elements of l_i. The choice of predicates by the knowledge engineer will impact upon the states of $\mathcal{E}$ that can be represented. Of course, the choice of predicates made by the knowledge engineer is not independent of the representation of the problem state used by the domain expert.

When a rule-based system is problem-solving, identifying that a goal state has been reached will be important for determining that a meaningful advancement in solving the problem has occurred. When the knowledge engineer specifies the structure of the problem that is to be solved (in terms of the subproblems it is decomposed into, the goal states for each subproblem, and the predicates to be used by the facts present in WM), the knowledge engineer must also indicate which of the predicates

are to be used to identify the goal states for each subproblem: these predicates are called *end predicates*, and the set of end predicates for a subproblem SP_t is denoted by Z_t. Each goal state g_i for a subproblem SP_t is identified by a conjunction of end predicates; thus when a rule-based system is problem solving, we can determine if a goal state has been reached.

Definition 5 (Logical Completion) *A logical completion for SP_t is a conjunction of selected predicates from Z_t, denoting a state which corresponds to a goal for SP_t.*

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. When the $r_i \in U$ fire, a goal state of the problem being solved has been reached. If we consider the beam analysis phase used by the Blackbox Expert as one of the subproblems of Blackbox, then a logical completion for beam analysis would be `GMAP_B` $\wedge$ `BALL`. The semantics attached to the goal state identified by `GMAP_B` $\wedge$ `BALL` is that a ball has been located in one of the Blackbox grid squares, and that grid square has been updated.

Our model for rule-based systems is intended to capture the structure of a rule base as chains of dependent rules. In our model, each path depicts a structural component of the rule base. Our intent is to construct chains of inter-dependent rules that advance the state of the problem being solved from one goal state to another, where goal states are identified by logical completions. The basic unit that will be used to construct paths will be the rule; thus, we must formally specify what is meant by a rule in our model. We will then examine how the intuitive notion of one rule being dependent upon another can be used to construct chains of rules that are inter-dependent. As our model is to capture sequences of rules that will fire when the rule-based system is problem-solving, we formally define when a rule is enabled by the rules that precede it in the sequence; this leads us to our definition for a path.

4.1.1 Abstract Rules

There are many different rule languages that have been reported in the literature [80]. Each language has its own syntax and provides different features: rule languages can be restrictive, allowing for rules that strictly follow the syntax of horn clauses and restrict the use of variables; or be flexible, permitting rules to contain complex

procedural code, and permit free use of variable expressions. In order for our model to be applicable to many rule-based systems, we must specify the exact form of rules to be used in the model, and develop methods for handling cases where the rule language used to code rules allows the use of constructs that do not match the form of the rules in our model. In order to reduce the complexity of our model, we specify rules that have a simple form: rules are modeled by considering only the facts they use and produce, in terms of the predicates appearing in expressions on their left hand side (LHS) and right hand side (RHS); thus, we require a method for abstracting the complex rules permitted by the different rule languages into the simple rules used in our model.

We refer to the rules used in our model as *abstract rules*. An abstract rule r_i is composed of an LHS and a RHS where: the LHS indicates the set of fact *templates*, denoted by $\mathcal{I}^{r_i}$, such that at least one fact *matching* each template must be present in WM for the rule to fire; the RHS indicates the set of facts that are asserted by r_i, denoted by $\mathcal{A}^{r_i}$. A template $t_i = \langle \Lambda_i, L_i \rangle$ is a predicate Λ_i and a list of variables L_i; thus, a template is a specification for the type of fact that must be present in the WM in order for a rule to become enabled to fire. A fact is said to match a template if there exists a most general unifier δ such that $\langle \Lambda_i, L_i \rangle \cdot \delta = \langle \lambda_i, l_i \rangle$. We also define a mapping function $\mho : A \rightarrow B$ where $A = \{\langle \Lambda_1, L_1 \rangle, \ldots \langle \Lambda_n, L_n \rangle\}$ is a set of templates or facts, $B = \{\lambda_1 \ldots \lambda_n\}$ is a set of predicates, and λ_i satisfies the specification for Λ_i. For simplicity, we will denote $\mho(\{\langle \Lambda_i, L_i \rangle\}) = \{\lambda_i\}$ by $\mho(\langle \Lambda_i, L_i \rangle) = \lambda_i$. The mapping function $\mho$ abstracts a template or fact, representing the template (fact) by the predicate it uses.

The abstraction provided by the mapping function $\mho$ permits the knowledge engineer to reduce the computational complexity in determining when a fact asserted by a rule matches a template contained on the LHS of another rule. In the case that the function $\mho$ is used to abstract a template or a fact, the knowledge engineer is indicating that the predicate used carries sufficient information to capture rule dependencies and variable bindings can be ignored; thus, the computation required for dependencies is reduced. If our model did not permit an abstraction of the facts used by the rule-based system, computing rule dependencies would require full unification

```
; 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 4.1: Sample CLIPS Rule

and constructing sequences of dependent rules would require checking every possible state of the system; this would be computationally intractable for complex rule-based systems. Of course, it is the responsibility of the knowledge engineer to select the predicates to be used in facts to ensure that they capture rule dependencies with the degree of accuracy that is required.

A sample rule (`Ball-Certain`) from the rule base of the Blackbox Expert is shown in Figure 4.1. Table 4.1 lists the predicates and pre-defined functions used by `Ball-Certain` and the other rules from the Blackbox Expert's rule base that will be used for example purposes. `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 square is already certain, in which case no action is needed. Otherwise, the square is made certain and, if a conflict exists, a fact using the predicate `RMC_B` is asserted indicating it can be resolved.

In the case that a rule contains conditionals and pre-defined functions on its RHS, the rule will be modeled by several abstract rules. Pre-defined functions represent an access to the Current Hypotheses module, which is modeled as an indirect access to WM; thus, each function will have a predicate associated with it (see Table 4.1). The conditionals on the RHS of a rule indicate that different actions can occur when the

USER FUNCTION	Interpretation	Associated Predicate
iscertain	check the certainty of a square	GMAP_CERT
setcertain	set a square certain	GMAP_CERT_B
status	check the contents of a grid square	GMAP

PREDICATE	Interpretation
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
GMAP	Access to the contents of a grid square
GMAP_B	Ball location on the grid
GMAP_E	Empty 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 4.1: Predicates and User Defined Function for Blackbox

rule fires; thus, we will create an abstract rule to capture each of the different actions that can occur when the rule fires. The abstract rules will capture the templates specified on the LHS of the rule and the facts asserted on the RHS of rule. The predicate associated with the test used in the condition that was on the RHS of the rule is placed on the LHS of the abstract rules.

In the case of our sample rule, `Ball-Certain` contains 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. `Ball-Certain` will be resented by two abstract rules `Ball-Certain%1`, and `Ball-Certain%2`. `Ball-Certain%1` will update the grid to indicate that a ball in a particular grid square is to be considered a certain ball, a conflict is discovered, and the conflict is to be resolved. $\mho(\mathcal{I}^{\texttt{Ball-Certain\%1}})$ = {`GMAP, GMAP_CERT, CERTAIN_BALLS, BALL_CERTAIN`}, and $\mho(\mathcal{A}^{\texttt{Ball-Certain\%1}})$ = {`GMAP_CERT_B, RMC_B, CERTAIN_BALLS`}. `Ball-Certain%2` will update the grid to indicate that a ball in a particular square is to be considered a certain ball. $\mho(\mathcal{I}^{\texttt{Ball-Certain\%2}})$ = {`GMAP, GMAP_CERT, CERTAIN_BALLS, BALL_CERTAIN`}, and $\mho(\mathcal{A}^{\texttt{Ball-Certain\%2}})$ = {`GMAP_CERT_B, CERTAIN_BALLS`}.

A set of abstract rules from the Blackbox Expert's rule base are shown in Table 4.2. When Path Hunter is creating abstract rules, it is possible that two abstract rules r_i and r_j, will have the same LHS and RHS: $\mho(\mathcal{A}^{r_i}) = \mho(\mathcal{A}^{r_j})$, and $\mho(\mathcal{I}^{r_i}) = \mho(\mathcal{I}^{r_j})$. This represents the case where there are two rules which are part of the rule base

that examine and produce facts using the same set of predicates: thus, in abstraction these two rules perform the same function. In the case of Blackbox, this arises due to the symmetry of the problem. Rules r_i and r_j form an equivalence class. Path Hunter will recognize that a group of rules forms an equivalence class, and uses equivalence classes when analyzing the structure of a rule base. Rules `RA-14-Right%1` and `RA-14-Left%1`, shown in Table 4.2, form an equivalence class called `RA-14-Class%1`.

The notion of an end predicate allows us to identify a set of rules in a task that will be instrumental in reaching a goal state of the subproblem. We say that a rule in a task is an *end rule* if every fact it asserts uses an end predicate. Formally, the set of end rules for a task T_t is $R_t^e = \{r_i \mid \forall \langle \lambda_i, l_i \rangle \in \mathcal{A}^{r_i},\ \lambda_i \in Z_t\}$. End rules must become enabled to fire in order for the rule-based system to reach a goal state, and end rules will play an important role when constructing paths.

4.1.2 Constructing Sequences of Inter-Dependent Rules

We will explore methods for constructing sequences of rules using a notion of rule dependency. In order for the sequences we construct to be useful for our purpose, we require that they have the following three properties: the sequences we construct must terminate when a goal state is reached; sequences of rules constructed for reaching a goal state must be "continuous", that is, they must not stop before a goal state is reached; and each sequence must be 'unambiguous', that is, the sequence represents only one way to reach a single goal state.

We now turn our attention to the types of "dependency" that can exist between two rules. Intuitively, we say that one rule is dependent upon another if the action taken by one rule facilitates the other rule to become fire-able. The simplest form of dependency exists when one rule asserts a fact that is required by the LHS of another rule.

Definition 6 (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 $\langle \lambda_i, l_i \rangle$ which uses a predicate that satisfies the predicate specified by a template in the LHS of r_j, with the constraint that $\lambda_i \notin Z_t$. Formally:* $r_i \prec r_j \equiv (\exists \langle \lambda_i, l_i \rangle \in \mathcal{A}^{r_i},\ \lambda_i \notin Z_t) \Rightarrow (\exists \langle \Lambda_i, L_i \rangle \in \mathcal{I}^{r_j},\ \mho(\langle \Lambda_i, L_i \rangle) = \lambda_i)$

The condition $\lambda_i \notin Z_t$ is placed on the depends upon relation in order to restrict this

r_i	$\mho(\mathcal{I}^{r_i})$	$\mho(\mathcal{A}^{r_i})$	*Comment*
RA-14-Right%1	{GMAP, GRIDSIZE, SHOT_RECORD}	{RA-14}	Indicate the occurrence of a specific configuration on the Blackbox grid.
RA-14-Left%1	{GMAP, GRIDSIZE, SHOT_RECORD}	{RA-14}	Indicate the occurrence of a specific configuration on the Blackbox grid.
RA-14-Prep%1	{RA-14}	{P_BALL, BLANK_GRID, BALL_CERTAIN}	Place a ball, mark a location as empty, and indicate that the Ball is a certain ball.
RA-14-Jprep%1	{RA-14}	{P_BALL, BLANK_GRID}	Place a ball, and mark a location as empty.
Place-Ball%2	{GMAP, GMAP_CERT, P_BALL}	{GMAP_B}	Update the grid to indicate that a ball has been located.
Place-Empty%1	{GMAP, GMAP_CERT, BLANK_GRID}	{GMAP_E}	Update the grid to indicate that an empty grid square has been located.

Table 4.2: Example Rule Set

relationship to rules that assert facts which do not identify goal states. Two examples of the depends upon relation exist between RA-14-Right%1, Right-14-Left%1, and RA-14-Prep%1 (shown in Figure 4.2). More precisely, RA-14-Right%1 $\prec$ RA-14-Prep%1 and RA-14-Left%1 $\prec$ RA-14-Prep%1.

When constructing chains of inter-dependent rules in a task, we group rules according to the dependency relationships in which they participate. Thus, for any rule in a task we desire the ability to identify those rules which it depends upon.

Definition 7 (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 $(\forall r_i \in T_i)(r_i \prec r_j \Rightarrow r_i \in V)$.*

For RA-14-Prep%1 in Table 4.2, V = {RA-14-Right%1, Right-14-Left%1}.

Having defined the depends upon relationship between two rules, we now consider grouping rules to form *continuous* sequences. We can ensure that a continuous sequence of rules is captured by defining a notion of *closure* on a set of rules for the depends upon relationship. When considering a set of rules Φ, we say that Φ is closed for depends upon if every rule in Φ asserts a fact either using an end predicate, or a predicate matching a template on the LHS of another rule in Φ. When a set of rules Φ is closed for depends upon, every rule in Φ asserting facts using a non-end predicate participates in depends upon relationships with the other rules in Φ; thus, Φ captures a continuous rule execution sequence.

Definition 8 (Dependency Closure) *A set of rules Φ is closed for the relation depends upon when every rule $r_i \in \Phi$ asserts facts where either $\lambda_k \in Z_t$, or the fact matches a template on the LHS of some $r_j \in \Phi$. Formally,* $(\forall r_i \in \Phi)(\exists r_j \in \Phi)$ *such that* $(\forall \langle \lambda_k, l_k \rangle \in \mathcal{A}^{r_i})((\exists \langle \Lambda_k, L_k \rangle \in \mathcal{I}^{r_j},\ \mho(\langle \Lambda_k, L_k \rangle) = \lambda_k) \vee (\lambda_k \in Z_t))$.

Given that GMAP_B, GMAP_E, CERTAIN_BALLS, and GMAP_CERT_B are end predicates we can determine that the set Φ = {RA-14-Right%1, RA-14-Prep%1, RA-14-Jprep%1, Place-Ball%2, Place-Empty%1, BallCertain%1} is closed for depends upon.

Although the set Φ is closed under depends upon, it is *ambiguous.* We say that a sequence or rules is ambiguous if it represents more than one way to reach a single goal, or represents ways in which more than one goal can be reached. When RA-14-Right%1 fires asserting a fact using RA-14, both RA-14-Jprep%1 and RA-14-Prep%1 can follow RA-14-Right%1 in the rule sequence. RA-14-Prep%1 asserts three predicates allowing Place-Ball%2, Place-Empty%1, and BallCertain%1 to follow it in the rule sequence, and RA-14-Jprep%1 asserts two predicates allowing Place-Ball%2, and Place-Empty%1 to follow it in the rule sequence. If the facts asserted by Place-Ball%2, Place-Empty%1, and BallCertain%1 form a logical completion and the facts asserted by Place-Ball%2, Place-Empty%1 also form a logical completion, then the rule sequence in Φ is ambiguous because two different goal states may be reached by the rule sequence. We can also demonstrate the case where a rule sequence containing two different groups of rules for reaching the same goal state can be constructed, which is also ambiguous. In order to remove these ambiguities from rule sequences, we introduce a restriction called *singular consumption.* The restriction of singular consumption on Φ ensures that every predicate that is asserted by a rule in Φ participates in exactly one depends upon relationship.

Definition 9 (Singular Consumption) *A set of rules Φ is singularly consumed if every fact that is asserted by a rule r_i is matched by a template of exactly one rule r_j where $r_i \prec r_j$. Formally, Φ is singularly consumed iff,* $(\forall r_i, r_j \in \Phi,\ r_i \prec r_j)$:

$$(\forall \langle \lambda_k, l_k \rangle \in \mathcal{A}^{r_i})(if\ (\exists \langle \Lambda_k, L_k \rangle \in \mathcal{I}^{r_j},\ \mho(\langle \Lambda_k, L_k \rangle) = \lambda_k) \Rightarrow (\nexists r_l \in \Phi,\ r_i \prec r_l,\ \langle \Lambda_l, L_l \rangle \in \mathcal{I}^{r_l},\ \mho(\langle \Lambda_l, L_l \rangle) = \lambda_k))$$

The set Φ = {RA-14-Right%1, RA-14-Prep%1, Place-Ball%2, Place-Empty%1, BallCertain%1} is singularly consumed. Thus, Φ contains an unambiguous sequence of rules for advancing the state of a problem to a particular goal state.

4.1.3 Enabling a Rule

The rule sequences we have considered thus far are not sufficient for our purposes because they do not consider the facts that are needed to enable each rule in the sequence to fire. 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 $(\forall r_i \in W)(\exists \langle \lambda_i, l_i \rangle \in \mathcal{A}^{r_i})$ such that $(\nexists r_k \in W,\ \langle \lambda_i, l_p \rangle \in \mathcal{A}^{r_k})$
- 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:

$$(\forall \langle \Lambda_i, L_i \rangle \in \mathcal{I}^{r_j})((\exists r_k \in V,\ \langle \lambda_i, l_i \rangle \in \mathcal{A}^{r_k},\ \mho(\langle \Lambda_i, L_i \rangle) = \lambda_i) \Rightarrow (\exists r_i \in W,\ \langle \lambda_i, l_i \rangle \in \mathcal{A}^{r_i},\ \mho(\langle \Lambda_i, L_i \rangle) = \lambda_i))$$

Definition 10 (Enablement) *A set of rules W enables a rule r_j iff W is a minimal set of rules that asserts 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 $r_j \in T_i$ and $V \rightarrow r_j$, $W \hookrightarrow r_j$ iff: $W \subseteq V$, and W is both minimal and maximal.*

For `RA-14-Prep%1` in Table 4.2 there are two enabling-sets: $W_1 = \{$`RA-14-Right%1`$\}$, and $W_2 = \{$`RA-14-Left%1`$\}$.

4.1.4 The Path

We consider a rule-base to be composed of a set of chains of inter-dependent rules called paths. A path in a rule base must identify a sequence of rule firings that can occur when the rule-based system is solving a subproblem, advancing the state of the problem to be solved from one goal state to another. Each path is composed of a sequence of rules that depend upon each other. Paths must capture continuous rule sequences, and they must be unambiguous. The set of rules comprising a path must

be defined such that each rule in the path is enabled by a subset of the rules contained in the path. The rules in each path must assert facts using all the predicates of a logical completion for a subproblem.

Definition 11 (Path) *P_k^t, a path k in task T_t is a partially-ordered set of rules $\langle \Phi, \pi \rangle$.*

Φ *is a set of rules* $\{r_1, r_2 \ldots r_n\}$ *with* $r_i \in T_t$ *such that,*
$(\exists U \subseteq \Phi,\ SP_t \rightsquigarrow U)$,
$(\forall r_i \in \Phi)(\exists W \hookrightarrow r_i, W \subset \Phi)$,
Φ *is closed under depends upon, and*
Φ *is singularly consumed.*

π *is a partial order indicating which rules in path* P_k^t *depend upon others;*
$(\forall r_i \in \Phi,\ r_i\ \pi\ r_j \Rightarrow r_i \prec r_j)$.

A path formed by the rules in Table 4.2 as found by Path Hunter is shown in Figure 4.2. This path represents the combined actions of five rules, depicting a meaningful advancement of the solution to the Blackbox puzzle: upon examining the evidence (beam entry and exit points) currently available, select the actions of placing a ball on the grid and marking a grid square as empty, and the outcome is that the ball is placed and the location is marked as empty. The logical completion that is asserted by this path is `GMAP_E` $\wedge$ `GMAP_B` $\wedge$ `CERTAIN_BALLS` $\wedge$ `GMAP_CERT_B`. The path shown in Figure 4.2 contains an equivalence class: `RA-14-Class%1`. Thus, the path shown in Figure 4.2 represents *two* paths that can be observed when the Blackbox Expert's rule base is executed: one path starting with `RA-14-Right%1`, and one path starting with `RA-14-Left%1`.

The path shown in Figure 4.2 is not a simple linear sequence of rules; this is unlike a linear sequence of statements that would be obtained when considering the execution paths in a traditional imperative program. This is not surprising because, with rule-based systems, we are not concerned with the actual sequences of rule firings that occur at run time, only rule dependencies. The sequence of statements that are considered to be an execution path of an imperative program form a total ordering, while the dependencies between the rules in a rule base form a partial ordering; hence, paths in a rule base are not linear sequences of rules.

We believe that our definition for path is appropriate for our objective: to capture the structure of a rule base as chains of inter-dependent rules (a path). Each path is

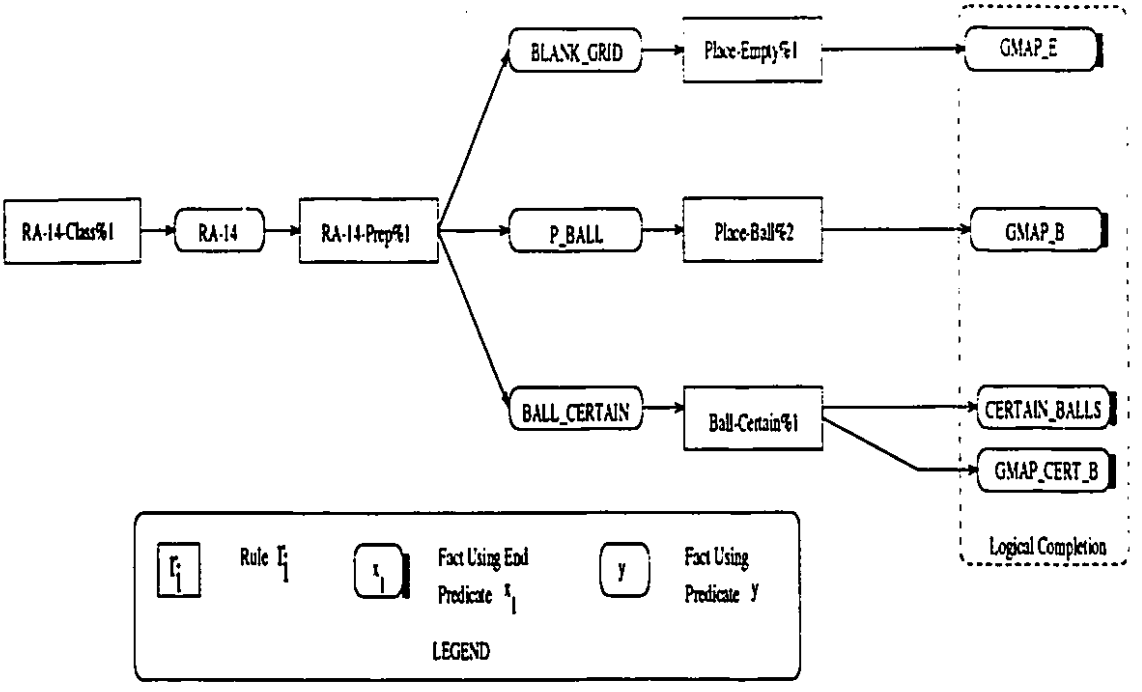


Figure 4.2: An Example Path

intended to depict a structural component of the rule base that advances the state of the problem being solved from one goal state to another, where goal states are identified by logical completions. We can show that each path contains all the rules that are needed to perform the computation required to advance the state of the problem being solved in a meaningful way (a goal state is reached). We first consider the inclusion of individual rules into a path based upon our depends upon relation, and then conclude that paths are *complete* with respect to the logical completion asserted by the rules in the path.

Lemma 1 *If there is at least one rule that asserts a fact* $\langle \lambda_i, l_i \rangle$, $\lambda_i \notin Z_t$, *and a path* P_k^t *contains a rule* r_i *where* $\langle \Lambda_i, L_i \rangle \in \mathcal{I}^{r_i}$, *and* $\mho(\langle \Lambda_i, L_i \rangle) = \lambda_i$, *then* $(\exists r_j \in \Phi,\ \langle \lambda_i, l_i \rangle \in \mathcal{A}^{r_j})$.

Proof We will assume that r_i is in path P_k^t, $\langle \Lambda_i, L_i \rangle \in \mathcal{I}^{r_i}$, $(\nexists r_k \in \Phi,\ \langle \lambda_i, l_i \rangle \in \mathcal{A}^{r_k},\ \mho(\langle \Lambda_i, L_i \rangle) = \lambda_i)$, and $(\exists r_j,\ \langle \lambda_p, l_p \rangle \in \mathcal{A}^{r_j},\ \mho(\langle \Lambda_p, L_p \rangle) = \lambda_i,\ \lambda_i \notin Z_m)$. Then $W, W \hookrightarrow r_i$ in path P_k^t is not maximal; thus, $(\exists r_k \in \Phi,\ \langle \lambda_i, l_i \rangle \in \mathcal{A}^{r_k},\ \mho(\langle \Lambda_i, L_i \rangle) = \lambda_i)$.

Corollary 1 (Completeness) *A path will contain all the rules* $r_i \in U$, $SP_t \rightsquigarrow U$ *as well as all the rules present in the rule base that enable the* $r_i \in U$.

Thus, paths will group together all the rules that are required to fire so as to advance the state of the problem being solved from one goal state to another.

4.2 Path Hunter

We have embodied our formal model for the structure of a rule base into Path Hunter, our rule base analysis tool. Given a list of the logical completions for a problem, Path Hunter will analyze a CLIPS rule base to determine the paths it contains. Several issues arise in the construction of the Path Hunter algorithm: analyzing a rule base to determine all the paths it contains can become too expensive to be practical, and we must be certain that the algorithm used to search for paths is sufficiently exhaustive to guarantee that no paths can remain undiscovered. Thus, we must employ methods that are economical in searching for paths, and we must show that the algorithm used to search for paths will discover all the paths that exist in a rule base.

The cost of searching for all the paths in a rule base can become prohibitive due to either the complexity of the algorithm used to construct each path, or due to the size of the space that must be searched to discover all the paths that exist in a rule base. Previous attempts by researchers in minimizing the cost of analyzing the structure of a rule base have focused on the cost of constructing each path, assuming that the size of the space that must be searched to discover all the paths is determined by the rules found in the rule base [79, 77]. Thus, if the size of the space as defined by the rules in the rule base was too large, these methods would be unable to analyze the structure of rule base (a combinatorial explosion in the size of the search space occurs). We believe that our model is unique because it permits the knowledge engineer to control the size of the search space, avoiding combinatorial explosions. Using our model, the cost of constructing each path is also minimized due to our method for the abstraction of rules and facts.

In this section, we discuss the Path Hunter algorithm, shown in Figure 4.3. We show that Path Hunter will discover all the paths that exist in a rule base. We then show how the size of the space that must be searched by Path Hunter can be controlled by the knowledge engineer; thus, combinatorial explosions are avoided. Finally, we discuss the results from our analysis of the rule base of the Blackbox Expert.

4.2.1 Discovering Paths

The first step in the Path Hunter algorithm is to compute the set of reachable rules V and the enabling sets W's for each rule in the rule base. Then, Path Hunter will begin constructing sequences of inter-dependent rules called *fragments*, denoted by F_k^t.

Definition 12 (Fragment) *A fragment F_k^t in task t is a set of rules that meets all the requirements for a path, except that it may not be closed under depends upon, or the rules in the fragment may not assert a logical completion.*

An example of a fragment that is not closed is shown in Figure 4.4. After creating all fragments, Path Hunter will *merge* fragments until a fragment is created that is closed for depends upon. The merge operation on two fragments is a union of the set of rules in each fragment, which forms another fragment. Two fragments may be merged if and only if they pass several tests to determine if the rule sequence that would result would also be a fragment. Once a fragment has been constructed that is closed for depends upon, it is tested to determine if the rules it contains assert a logical completion. If the rules in the fragment assert a logical completion, the fragment is accepted as a valid path, else it is discarded.

Three fragments constructed by Path Hunter that are not closed are shown in Figure 4.5. Path Hunter constructs each fragment starting with an end rule; Path hunter selects an enabling set for the end rule, and then adds all the rules in the enabling set to the fragment. Path Hunter then selects an enabling set for one of the rules in the enabling set just added to the fragment, continuing the process until it encounters a rule that does not have an enabling set. When Path Hunter constructed Fragment B, shown in Figure 4.5, it selected the end rule `Place-Ball%2`, chose `RA-14-Prep%1` as `Place-Ball%2`'s enabling set, and then selected `RA-14-Class%1` as `RA-14-Prep%1`'s enabling set. `RA-14-Class%1`'s enabling set is empty; thus, Path Hunter determined that the end of Fragment B had been found.

The three fragments shown in Figure 4.5 were merged by Path Hunter to form the sample path given in Figure 4.2. During the merge phase, Path Hunter examines each fragment to determine if it is closed. If a fragment is not closed, Path Hunter will identify the *splice rules* in the fragment.

```
Program Path_Hunter                          {Find All Paths in Rule Base RB}
{
 build_fragment(r_i)                         {Build one fragment starting from r_i}
 {
  B_PF = r_i;                                {B_PF accumulates r_i in fragment}
  Get Next W ⟵ r_i;                          {Get a W for r_i}
  Mark W as Seen;
  Foreach r_j ∈ W Do                         {Get rest of fragment starting from r_j}
    B_PF = B_PF ∪ build_fragment(r_j);
  Return(B_PF)
 }

 {
  Foreach rule r_i ∈ RB Do                   {Find all rules that r_i depends upon}
   {                                         {Find all enabling sets for r_i}
    Find V such that V — r_i;
    Find All W such that W ⟵ r_i;
   }

  Foreach rule r_i ∈ RB Do                   {Generate all fragments for each end rule}
    If (r_i ∈ Z_i) then
      Repeat
      {
        PF = build_fragment(r_i);
        Record PF;
      }
      Until (Seen All Combinations of W's starting from r_i)
  Foreach Fragment FR_i Do                   {Find all Splice-Rules in the fragment}
    Find All splice-rules SPR_j ∈ FR_i;
  Foreach Fragment FR_i Do                   {Merge fragments to form paths}
  {
    PP = FR_i;                               {PP accumulates potential path}
    Repeat
    {
      While Not_Closed(PP) do
      {
        Get next FR_j Matching SPR_j ∈ PP;
        Mark FR_j as Seen;
        If (Common_Origin(FR_j, PP) AND
            NOT_Split_Tail(FR_j, PP)) then
          PP = Merge(PP, FR_j);              {Combine potential path and fragment}
      }
      If Asserts_Logical_Completion(PP) then
        Record PP;                           {PP is a valid path}
    }
    Until (Seen All Combinations of FR_j's for FR_i)
  }
 }
}
```

Figure 4.3: Path Hunter Algorithm

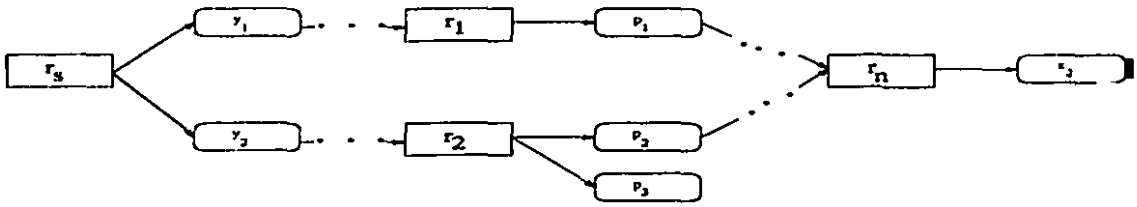


Figure 4.4: An Example Fragment

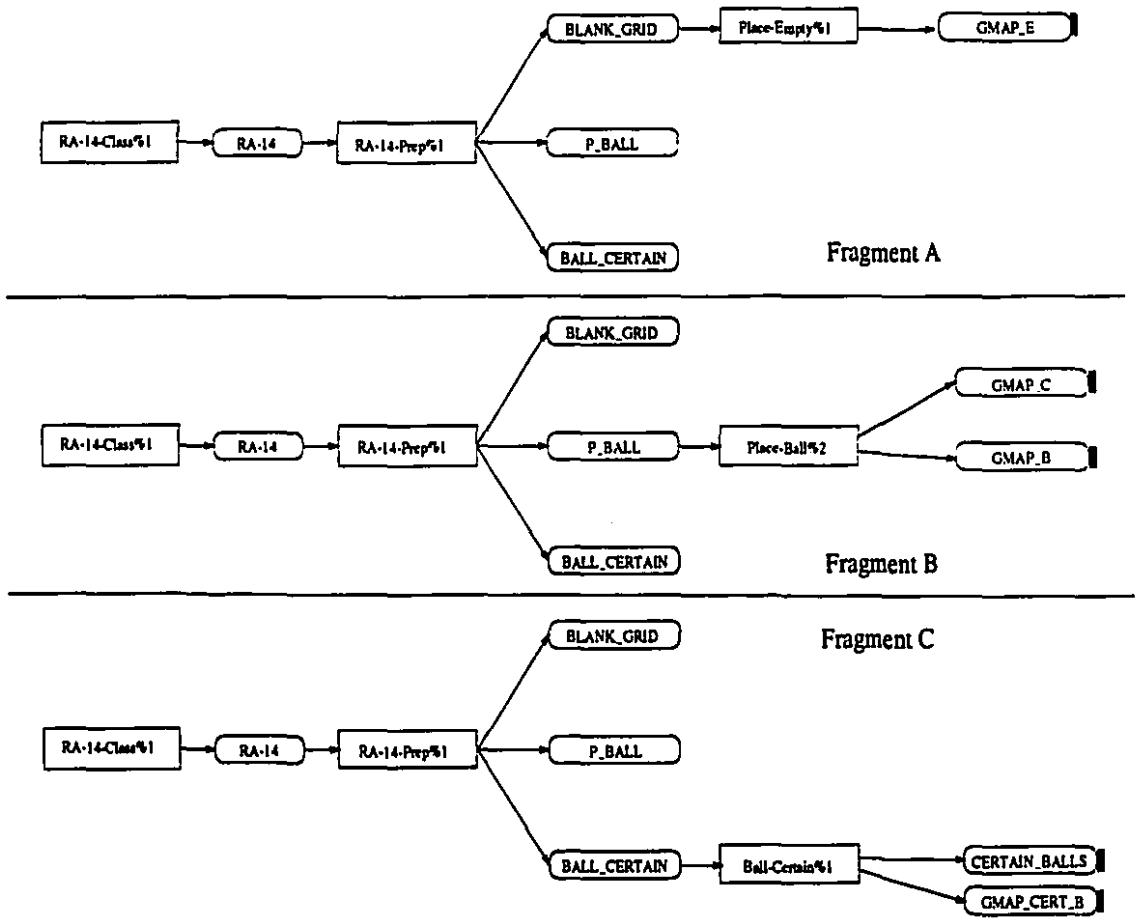


Figure 4.5: Constructing Paths from Fragments

Definition 13 (Splice Rule) *A rule r_i is said to be a splice rule in fragment F_k^t, if it asserts at least one fact using a non end predicate which is not specified by a template on the LHS of any other rule in the fragment. Formally, r_i is a splice rule in F_k^t iff:*

$$\exists(\langle \lambda_k, l_k \rangle \in \mathcal{A}^{r_i}) \wedge (\nexists(r_j \in \Phi),\ \langle \Lambda_k, L_k \rangle \in I^{r_j},\ \mho(\langle \Lambda_k, L_k \rangle) = \lambda_k)$$

`RA-14-Prep%1` is a splice rule. Using splice rules, Path Hunter determines the fragments that could result in fragments that would be closed if merged. However, before Path Hunter merges two fragments, it performs two additional tests to determine if the rule sequence formed by merging the fragments would be a valid fragment: the *common-origin* test, and the *split-tail* test (see Figure 4.3).

When considering if the Path Hunter algorithm finds all paths that are present in a rule base, we must examine two issues: does the algorithm generate the fragments required to construct all the paths, and are all the appropriate combinations of fragments merged. When generating fragments, we must determine if all fragments can be constructed starting from the end rules. We will assume that it is possible to produce error free code that selects all combinations of enabling sets for a rule when generating fragments. When merging fragments, we must determine if the two tests used by Path Hunter permit the merging of only combinations of fragments that may form valid paths. We assume it is possible to produce error free code to determine all the fragments that are potential candidates for being merged.

We can show that by starting its search for paths with end rules, Path Hunter is guaranteed to find the fragments required to construct all the paths in the rule base. In order to show this, we prefer to view paths in an alternative manner. All paths can be placed into one of three categories: *single rule* paths, *single chain* paths, or *multiple chain* paths. A chain is a sequence or rules that depend upon each other: the first rule in the sequence has no enabling sets within the rule base, the last rule in the sequence is an end rule, and only one of the facts asserted by each rule in a chain matches a template on the LHS of another rule in the chain. Single rule paths contain only one rule, (see Figure 4.6, ignore shaded region). An example of a single chain path is shown in Figure 4.7 (ignore shaded region), and an example of a multiple chain path is shown in Figure 4.8 (ignore shaded region). By categorizing paths in this manner, we will show that every path must contain at least one end rule. When

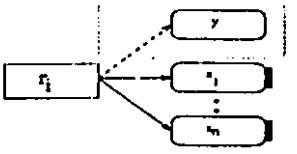


Figure 4.6: Single Rule Paths

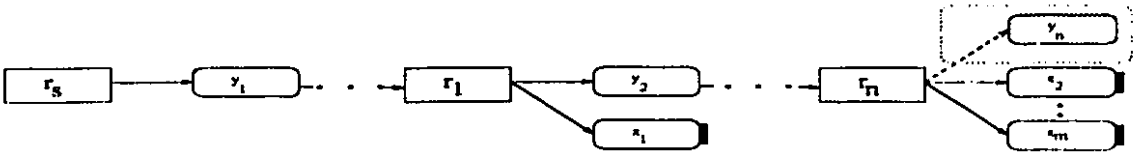


Figure 4.7: Single Chain Paths

proving that each path must contain an end rule, we will also show that each chain in a path will terminate with an end rule. We conclude that every fragment required to construct all the paths in a rule base can be generated starting with the end rules because fragments can be categorized in the same manner that we have categorized paths.

Theorem 1 *Every path P_k^t must contain at least one end rule.*

Proof To show that every path must contain at least one end rule we first show that all paths are finite, and then consider the case of the single rule path, the single chain path, and the multiple chain path. We can see that all paths are finite because the number of rules in any path is bounded by the number of rules in the rule base, which is of course finite.

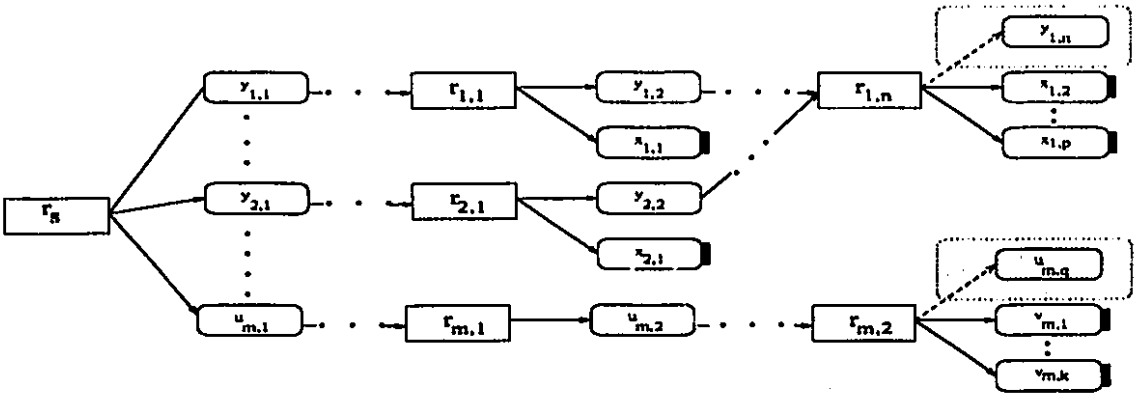


Figure 4.8: Multiple Chain Paths

Case 1: We will assume that Figure 4.6 is an example of a single rule path, that a logical completion for T_t is $\{r_1, \ldots, r_n\}$, but r_i is not an end rule because it asserts a fact using a predicate y, and $y \notin Z_t$. However, given these assumptions the path shown in Figure 4.6 is not a valid path because Φ is not closed. This contradicts our original assumption that Figure 4.6 depicts a single rule path; thus, r_i must be an end rule, and r_i cannot assert a fact using predicate y.

Case 2: We will assume that Figure 4.7 is an example of a single chain path. Single chain paths are finite; thus, there is a rule r_n that is the last rule in the path. The same argument used in case 1 is applied to r_n; thus, r_n is an end rule.

Case 3: We will assume that Figure 4.8 is an example of a multiple chain path. In the case of the multiple chain path, there will be a rule $r_{i,n}$ at the end of each chain. Again, the same argument used in case 1 can be applied to $r_{i,n}$; thus, $r_{i,n}$ is an end rule.

The proof for Theorem 1 allows us to draw two conclusions about the structure of paths.

Corollary 2 *Every chain in a path terminates with an end rule.*

Corollary 3 (End-Rule Complete) *All the paths that exist in a rule base can be found by starting the search with the end rules.*

Thus, the Path Hunter algorithm which starts its search with the end rules in the rule base and follows the chain of rule dependencies until it encounters a start rule will discover the fragments required to construct all the paths in the rule base.

The merge phase of the Path Hunter algorithm is also important in establishing its correctness. Two fragments are candidates for being merged if they share a common splice rule. However, the two fragments are merged only if they differ in the rule that depends upon the splice rule: the LHS of the rule depending upon the splice rule in each fragment must contain a template specifying a different predicate from the set of predicates used by the facts asserted by the RHS of the splice rule, if the merge phase is to produce fragments that are closed for depends upon. As an example, Fragment A and Fragment B in Figure 4.5 contain the splice rule `RA-14-Prep%1`, but in Fragment A `RA-14-Prep%1` $\prec$ `Place-Empty%1`, and in Fragment B `RA-14-Prep%1` $\prec$ `Place-Ball%2`. When Fragment A and Fragment B are merged, `RA-14-Prep%1` will have only one fact on its RHS that uses a predicate that

is not specified by the LHS of any rule in Fragment A; thus, merging Fragment A and Fragment B produces a fragment that is closer to being closed for depends upon than either Fragment A or Fragment B.

Once two fragments have become candidates to be merged, we must consider the conditions that permit two fragments to be merged producing a rule sequence that is a valid fragment. We treat the *origin* of each fragment in a different manner than its *tail*. The origin of a fragment is the sequence of rules it contains before the splice rule, and the tail of a fragment is the sequence of rules it contains after the splice rule. Two fragments may be merged if and only if their origins are identical; Figure 4.9 shows the case where Fragment D and Fragment E are merged to form Fragment F, but they do not have identical origins; in the proof for theorem 2, we show that Fragment F is not a valid fragment. The first rule in the tail of two fragments that are candidates for being merged are always distinct. Two fragments may be merged only if the *remainders* of the tails in each fragment are identical.

Definition 14 (Remainder) *The remainders of the tails of two fragments is the sequence of rules starting with the first rule the tails have in common, ending with the end of the tail of each fragment.*

Figure 4.10 shows the case where Fragment G and Fragment H are merged to form Fragment I, but they do not have identical remainders; in the proof for theorem 3, we show that Fragment I is not a valid fragment.

Theorem 2 (Common-Origin) *Two fragments may be merged iff their origins are identical.*

Proof Let us assume that the origin of two fragments (Fragment D and Fragment E) differ by one rule as shown in Figure 4.9, but Fragment F formed by merging Fragment D and Fragment E is a valid Fragment. In Fragment D we have $r_s \prec r_a$, and in Fragment E we have $r_s \prec r_b$; thus in Fragment F we have $r_s \prec r_a$ and $r_s \prec r_b$, but $\mho(\mathcal{A}^{r_s}) = \{y_1\}$. Therefore, $\mho(\mathcal{A}^{r_s})$ is not singularly consumed, and Fragment F is not a valid fragment. This violates our assumption that the origin of two fragments may differ by one rule and still produce a valid fragment when merged; thus, two fragments can be merged to form a new fragment only if their origins are identical.

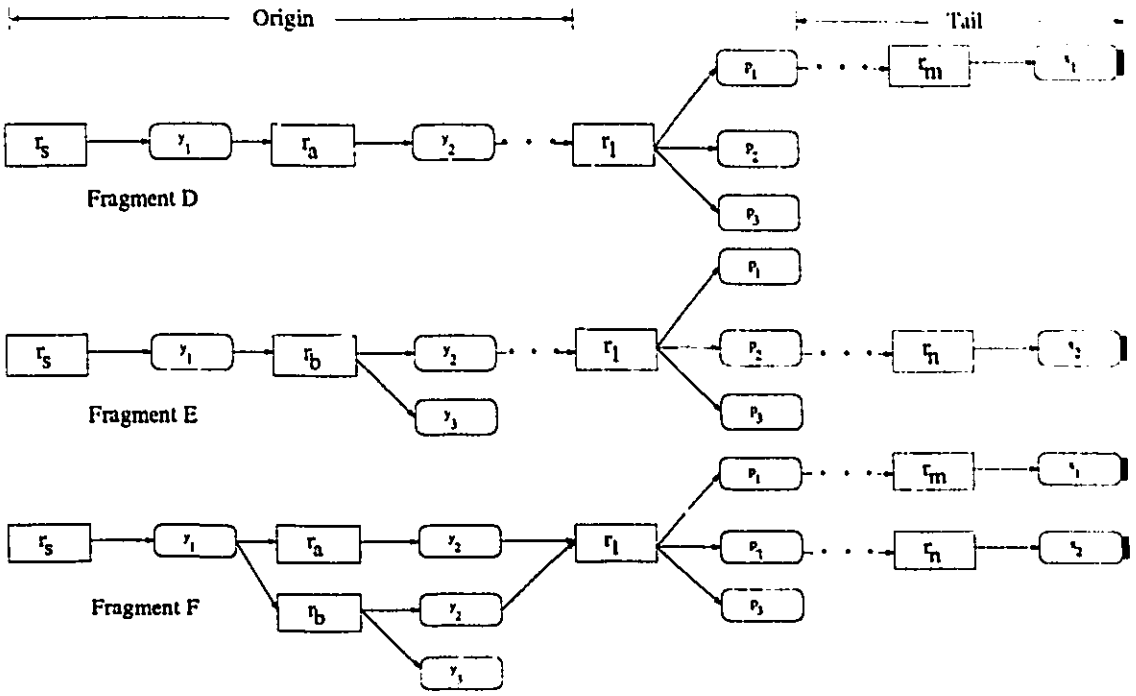


Figure 4.9: Merge: Non-Common Origin

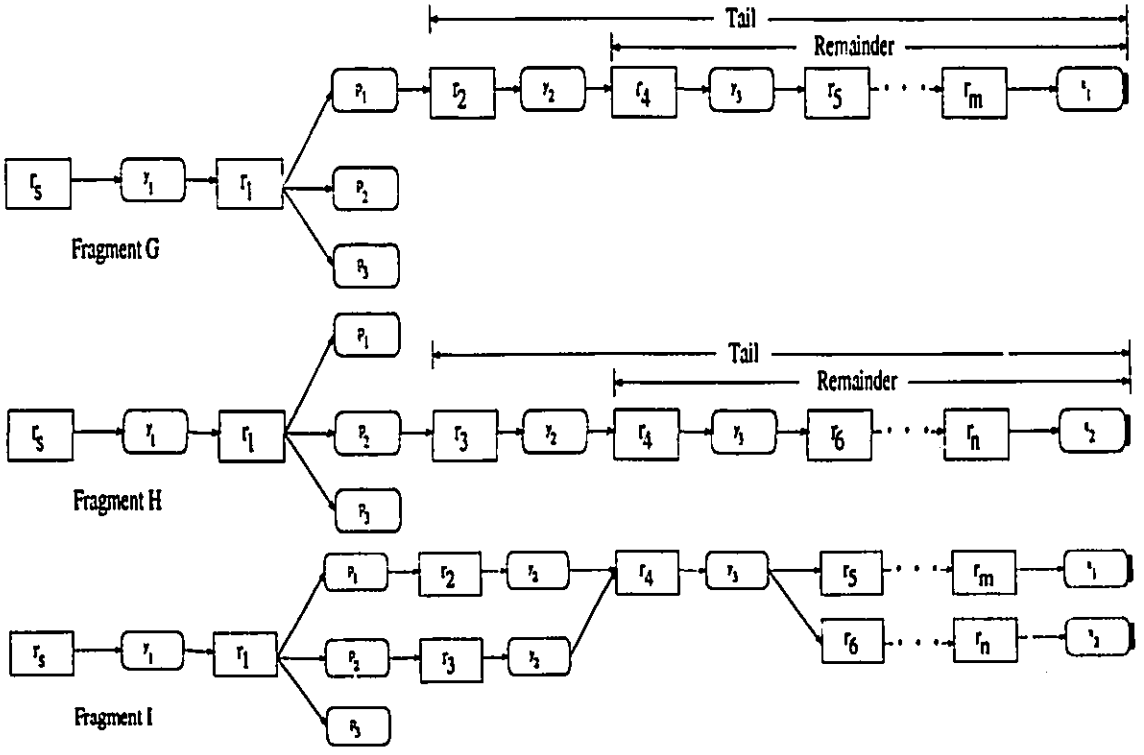


Figure 4.10: Merge: Non Identical Remainder

Theorem 3 (Split-Tail) *Two fragments may be merged iff their remainders are identical.*

Proof Let us assume that the remainder of two fragments (Fragment G and Fragment H) differ by one rule as shown in Figure 4.10, but Fragment I formed by merging Fragment G and Fragment H is a valid Fragment. In Fragment G we have $r_4 \prec r_5$, and in Fragment H we have $r_4 \prec r_6$; thus in Fragment I we have $r_4 \prec r_5$ and $r_4 \prec r_6$, but $\mho(\mathcal{A}^{r_4}) = \{y_3\}$. Therefore, $\mho(\mathcal{A}^{r_4})$ is not singularly consumed, and Fragment I is not a valid fragment. This violates our assumption that the remainder of two fragments may differ by one rule and still produce a valid fragment when merged; thus, two fragments can be merged to form a new fragment only if their remainders are identical.

Path Hunter will find all the paths that exist in a rule base. Corollary 3 assures us that Path Hunter will discover every fragment in a rule base by starting its search for fragments with end rules. Theorem 2 and Theorem 3 assures us that the Common-Origin and Split-Tail tests used in Path Hunter Algorithm (shown in Figure 4.3) do not permit two fragments to be merged that will not result in a valid fragment.

4.2.2 Controlling Combinatorial Explosion

There are two stages in the Path Hunter algorithm that are prone to combinatorial explosion: fragment generation, and the merging of fragments. During fragment generation, the number of fragments produced can become too large to be computationally tractable. During the merge phase, the number of combinations of fragments that must be merged to discover all the paths in the rule base can become unmanageable. In both these cases, the combinatorial explosion is controllable by the logical completions that have been specified by the knowledge engineer.

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 knowledge engineer 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

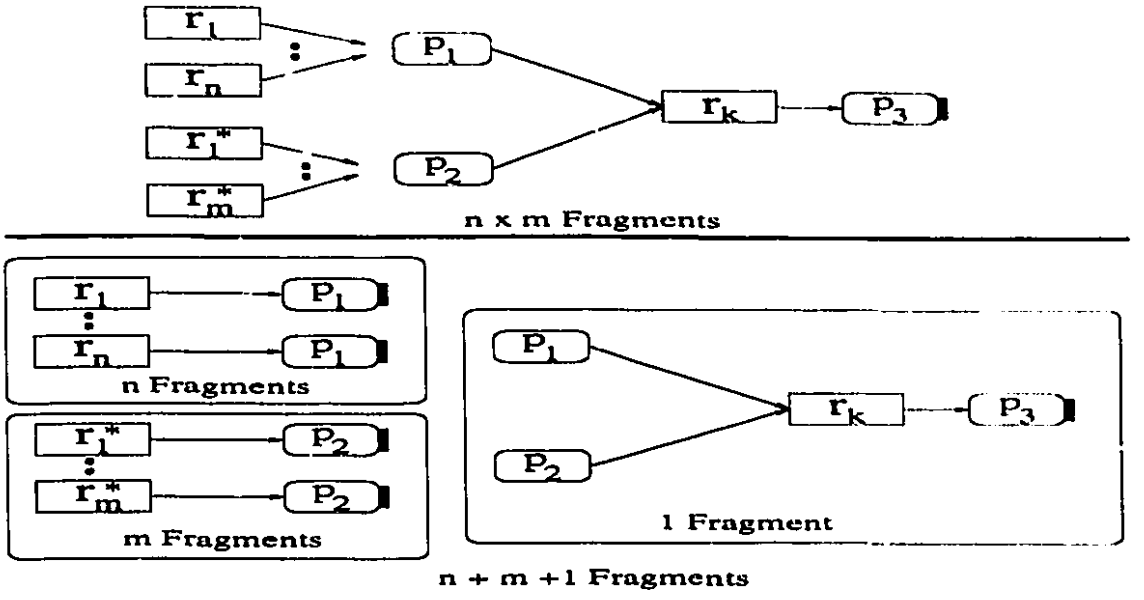


Figure 4.11: Combinatorial Explosion Generating Fragments

were to be created for the original logical completion.

An example of where combinatorial explosion occurs when fragments are generated is shown in Figure 4.11. Figure 4.11 shows the scenario where p_3 is a logical completion, r_k is an end rule, there are n rule sequences producing a fact using predicate p_1, and m rule sequences producing a fact using predicate p_2. Path Hunter will generate fragments starting from r_k, and it will generate $n \times m$ fragments: one fragment for each possible combination of rule sequences producing p_1 and p_2. If n and m are large, a combinatorial explosion in the number of fragments will occur. Based upon the number of rule sequences present in the rule base for producing facts using predicates p_1 and p_2, Path Hunter is indicating the importance of p_1 and p_2 in solving the problem; thus, p_1 and p_2 should be goal states. When the logical completions are changed to reflect the importance of p_1 and p_2, Path Hunter will generate $n + m + 1$ fragments, as shown in Figure 4.11.

An example of where combinatorial explosion occurs when fragments are merged is shown in Figure 4.12. Figure 4.12 shows the scenario where $p_1 \wedge p_2$ is a logical completion, there are n rule sequences starting from p_3 generating a fact using p_1, and m rule sequences starting from p_4 generating a fact using p_2. Path Hunter will generate n fragments starting with r_k and ending with p_1 as well as m fragments starting with r_k and ending with p_2. In the $m + n$ fragments, r_k is a splice rule, and

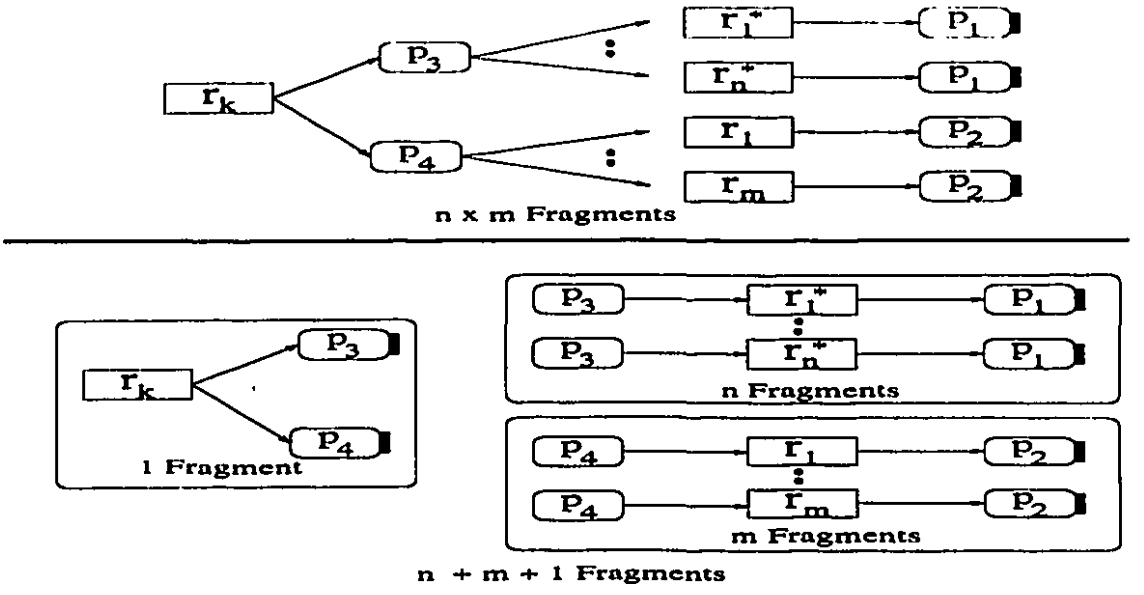


Figure 4.12: Combinatorial Explosion Merging Fragments

Path Hunter will attempt to merge the fragments in $n \times m$ combinations. If n and m are large, a combinatorial explosion will occur. Path Hunter, in performing its analysis, is indicating the importance of p_3 and p_4 in solving the problem; thus, p_3 and p_4 should represent a goal state; When the the logical completions are changed to reflect the importance of p_3 and p_4, Path Hunter will generate $n+m+1$ fragments, as shown in Figure 4.12

The knowledge engineer can control any combinatorial explosions that occur when using Path Hunter to analyze a rule base by introducing additional goal states. When new goal states are introduced, 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.

4.2.3 Analyzing the Blackbox Expert's Rule Base

Given the set of logical completions, Path Hunter has been used to analyze 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 as well as 170 rules not in any equivalence class. Path Hunter found 516 paths, and we believe this is reasonable given the complexity of the Blackbox puzzle and our

set of logical completions; there are rule bases which cause the generation of many thousands of paths from a small number of rules, as was demonstrated by the analysis of the ONCOCIN rule base [81]. The smallest paths consisted of a single rule; the deepest had a depth of 7 rules; the broadest had a breadth of 7 rules. The mean path depth was 4 rules, and the mean path breadth was 3.5 rules. The paths produced by Path Hunter have been manually verified by the knowledge engineer as being accurate and 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.

The process of applying Path Hunter to the Blackbox Expert's rule base identified an application for our structural model apart from our intended application of the path model to CDPS design. We have found that our structural model will also play an important role in the validation of rule-based systems [35, 38, 37]. Path Hunter identified various anomalies in the Blackbox Expert's rule base: the improper use of predicates, undesired dependencies 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 because the rule was dependent upon rules from one task, but the rule was performing actions that would reach a goal state from a different task. The use of Path Hunter to analyze the Blackbox Expert's rule base provided a method to validate the design of the rule base by indicating these inconsistencies and ambiguities.

An example of where a combinatorial explosion occurred when Path Hunter was analyzing the rule base of the Blackbox expert is shown in Figure 4.13. Figure 4.13 shows the structure of the paths that Path Hunter was attempting to produce when

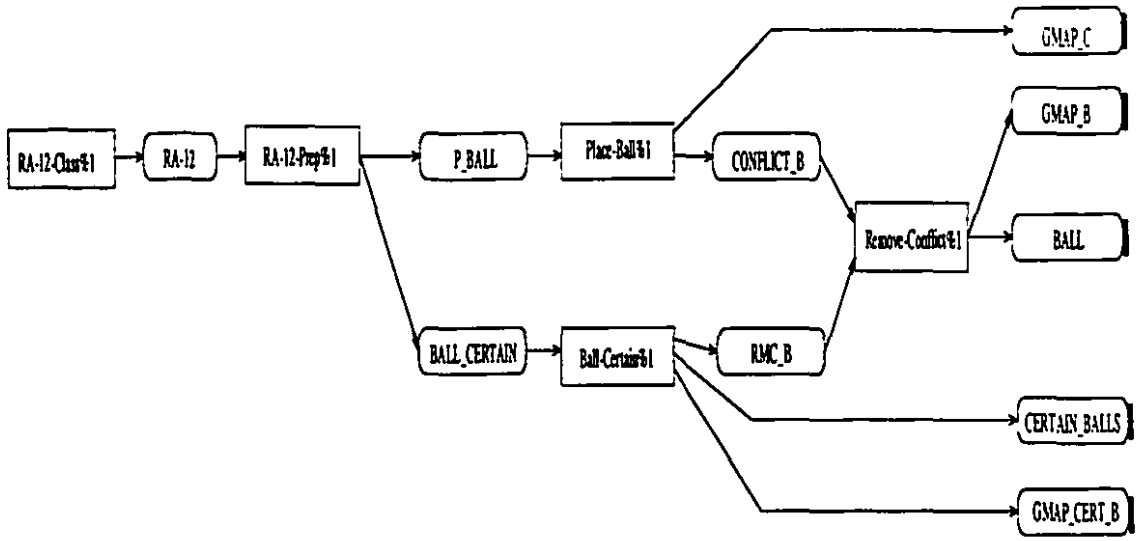


Figure 4.13: Controlling Combinatorial Explosion in Practice

GMAP_C ∧ GMAP_B ∧ BALL ∧ CERTAIN_BALLS ∧ GMAP_CERT_B was declared by the rule base designer as a logical completion. The predicate CONFLICT_B indicates that a conflict occurred when a ball was to be placed in one of the squares of the Blackbox grid, and RMC_B indicates that a conflict can be resolved. A combinatorial explosion resulted because there are many rule sequences in the Blackbox Expert's rule base for placing a ball as well as for determining that a conflict can be resolved. Path Hunter's analysis pointed out to the rule base designer that placing a ball or determining that a conflict could be resolved should be considered as goal states, and that the action of actually resolving a conflict should be treated separately from a conflicts creation or detection.

4.3 Conclusion

Our model for the internal structure of a rule-based system captures the structure of a rule base as chains of inter-dependent rules (paths). Each path depicts a structural component of the rule base that advances the state of the problem being solved from one goal state to another, where goal states are identified by logical completions. Our model also meets our three criteria,

- Our model is accurate. The path, as we have defined it, has served as a specification for the construction of the Path Hunter algorithm. The concepts of logical

completions, depends upon, enablement, singular consumption, and closure ensure that paths only represent sequences of rules that can occur at run-time, and this has been demonstrated by the use of Path Hunter on the Blackbox Expert.

- Paths are meaningful. Our structural model accounts for the structure of the problem being solved, providing a notion of meaningfulness for the actions taken by the rules in a path.

- Our structural model is computationally tractable. With our model, the knowledge engineer, using logical completions, can control any combinatorial explosion that can potentially occur when Path Hunter is generating the paths in a rule base. This was also demonstrated by the use of Path Hunter on the Blackbox Expert.

Chapter 5

Applying the Organization and Path Models

"The woods are lovely, dark and deep.
But I have promises to keep,
And miles to go before I sleep,
And miles to go before I sleep."
- Robert Frost, "Stopping by Woods on a Snowy Evening", 1923

Applying the organizational model and the path model to building CDPS enables us to consider both an inter-agent perspective and an intra-agent perspective; we focus on the use of these models in setting the data distribution in a CDPS. The organizational model is designed to capture the inter-agent perspective and the path model is applied to capture the intra-agent perspective. The organizational model specifies the interactions that occur between the rule-based systems in a CDPS, including the data items available to the rule-based systems. We apply the path model to capture the data items required by a rule-based system to produce specific results. Using the path model, the CDPS designer can determine the impact of the availability of a specific data item on the result produced by a rule-based system that is a member of a CDPS, aiding the CDPS designer to set the availability of specific data items (the windows specified using the organizational model), in order to improve the performance of the rule-based systems in a CDPS. Thus, when specifying the interactions that are part of the inter-agent perspective, the CDPS designer is guided by studying the intra-agent perspective.

When studying the intra-agent perspective, the CDPS designer is concerned with

determining how to set the availability of specific data items in order to improve the performance of the rule-based systems in a CDPS, or avoid serious failures. In determining how to set the availability of data items, the CDPS designer is faced with the following questions:

- what is the set of data items required by the rule-based systems to achieve each goal?
- what is the impact of a particular data item being unavailable on the ability of the rule-based system to achieve a goal?
- what is the direct impact of achieving each goal on the ability of the rule-based system to produce a result, or the quality of the result produced?
- what are the *interactions* that occur between goals as the data items available are reduced? That is, when a rule-based system cannot achieve a goal due to the unavailability of a particular data item, will it achieve an alternate goal, and if so, which goal is achieved as an alternate?
- in the case that a rule-based system achieves a goal when operating with a reduced number of data items (this goal may be an alternate goal achieved due to an interaction), is it desirable for the rule-based system to achieve that goal? That is, will the result produced by achieving that goal be acceptable?
- if a goal is achieved by a rule-based system when operating with a reduced number of data items, but the CDPS designer determines that it is not desirable for that goal to be achieved, how can the availability of data items be adjusted to avoid achievement of that goal? Which path is responsible for that goal being achieved, and which data items are required by the rules in that path to fire?

The understanding gained by the CDPS designer in studying intra-agent perspective to answer these questions will aid the CDPS designer to set the availability of specific data items in the CDPS. The CDPS designer can set the availability of data items to ensure that the goals that are most "important" are achieved; each goal can be assigned a priority when studying the intra-agent perspective to determine the impact of achieving that goal on the ability of the rule-based system to produce a result, or the quality of the result produced. The CDPS designer can also ensure that the achieving of goals that would produce unacceptable results is avoided, by using paths to identify the data items responsible for the achievement of those goals, and then adjusting the data distribution in the CDPS.

Applying the path model to study the intra-agent perspective is accomplished in two separate steps as follows:

step 1 determine the data items that are required by the rules in each path to achieve the goal identified by the logical completion asserted by the rules in that path. We determine data items required by analyzing each path to determine the data items required for all the rules in that path to fire.

step 2 determine the effect on the result produced when a goal is achieved; we determine this by monitoring the goals that are achieved by a rule-based system as it solves a set of test cases. Monitoring the goals that are achieved by a rule-based system permits us to study the goals achieved by a rule-based system as we reduce the data items available.

In this chapter, we show how the organizational model and the path model can be applied to aid the CDPS designer set the data distribution in a CDPS. We explain how to analyse paths to determine the data items that are required by the rules in each path to fire, and then present an algorithm for this analysis. We present a method for monitoring the goals that are achieved by a rule-based system, by determining when all the rules in a path have fired. We have constructed a tool, Path Tracer, embodying our method for monitoring the goals that are achieved by a rule-based system. We present results from an experiment to validate the accuracy of our method for monitoring goals achieved; in this experiment, Path Tracer is used to determine the goals that were achieved by the Blackbox Expert when it solved the set of test cases used for the functional validation experiment described in chapter 3. In order to demonstrate the use of our models (organizations and paths) and tools (Path Hunter and Path Tracer), we then present a case study in which we examine the intra-agent perspective for the Blackbox Expert and consider setting the data distribution in a CDPS for solving the Blackbox Puzzle.

5.1 Data Items Required

In this section, we first present an approach for analyzing the paths in a rule base to identify the data items required for all the rules in each path to fire, and then we present an algorithm for implementing this approach. We also show that the cost of analyzing the paths in a rule base using our algorithm grows linearly with an increase in the number of rules and paths in the rule base.

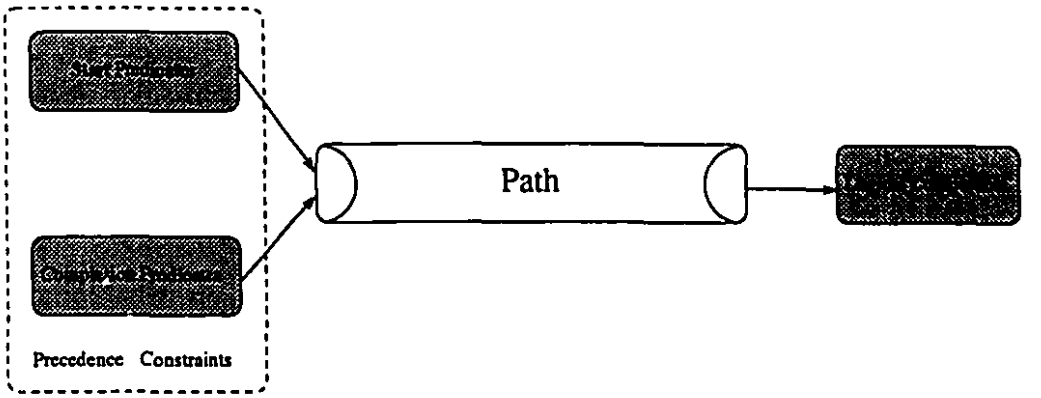


Figure 5.1: Data Requirements

5.1.1 Identifying Data Items Required

We can view a path as a set of rules that require certain facts to achieve a specific goal, see Figure 5.1. The facts required by the rules in a path are the precedence constraints for achieving the goal. There are two issues that must be tackled when using the path model to determine the facts required by a rule-based system: determining the precedence constraints required for a single rule in a path to fire, and determining when all the rules in a path will be able to fire.

The first rules that must fire in a path are called *start rules*. Start rules will then assert facts enabling other rules in the path. The start rules of a path, denoted by SR_k^l, is the set of rules $r_i \in \Phi$ where the templates in the LHS of r_i do not depend upon any other rule $r_j \in \Phi$; $SR_k^l = \{r_i \mid r_i \in \Phi, (\forall r_k \in \Phi,\ r_k \not\prec r_i)\}$. The start rule for the example path shown in Figure 4.2 is in fact an equivalence class $SR_{\texttt{example}}^{\texttt{Beam Analysis}} = \{\texttt{RA-14-Class\%1}\}$.

A *start set* of a path is a set of facts matching the LHS of all the start rules of that path. A start set of a path is identified by the set of *start templates*, denoted by ST_k^l, which is the union of the set of templates present on the LHS of each of the start rules of the path; $ST_k^l = \bigcup_{r_i \in SR_k^l} \mathcal{I}^{r_i}$. The set of predicates used in the start templates for a path, called the *start predicates*, indicate factors from P^l from which data items must be available to enable the start rules to fire, this set is given by $SP_k^l = \{\lambda_j \mid \forall(\Lambda_i, L_i) \in ST_k^l,\ \mho((\Lambda_i, L_i)) = \lambda_j\}$.

The facts required for all the rules in a path, except start rules, to fire is called a *completion set*. The completion set of a path is identified by the set of *completion tem-*

plates for a path, denoted by CT_k^t. The completion templates is the set of all templates present on the LHS of all the rules of the path, except start rules, which specify a fact that is not asserted by any rule in the path. Formally, $CT_k^t = \bigcup_{r_i \in (\Phi - SR_k^t)} (\mathcal{I}^{r_i} - PT_{i,k}^t)$. $PT_{i,k}^t$ is the set of templates from $\mathcal{I}^{r_i}$ that are matched by the facts asserted when the rules that enable r_i fire. $PT_{i,k}^t = \{\langle \Lambda_i, L_i \rangle \mid (\forall r_j \in W,\ W \hookrightarrow r_i)(\exists \delta,\ \langle \Lambda_i, L_i \rangle \cdot \delta = \langle \lambda_i, l_i \rangle,\ \langle \lambda_i, l_i \rangle \in \mathcal{A}^{r_j})\}$. The set of predicates used in the completion templates for a path, called the *completion predicates*, indicate the factors from P^I for which instances (data items) must be available so that all the rules in the path become enabled to fire, assuming the start rules have fired. Formally, the set of completion predicates is given by $CP_k^t = \{\lambda_j \mid \forall \langle \Lambda_i, L_i \rangle \in CT_k^t,\ \mho(\langle \Lambda_i, L_i \rangle) = \lambda_j\}$.

Now, let us consider the precedence constraints for a rule in a path. A rule in a path can fire when there are facts present in WM that satisfy all the templates in its LHS. Facts matching the LHS of a rule can be present in WM because they were placed into WM by other rules in the path, or because they were present in WM before the rules in the path fired. Given a path P_k^t, $r_i \in \Phi$, $r_i \notin SR^t$, and $W \hookrightarrow r_i$, then the set of completion templates for r_i is given by $CT_{i,k}^t = (\mathcal{I}^{r_i} - PT_{i,k}^t)$.

Lemma 2 *If facts matching the completion templates $CT_{i,k}^t$ of rule r_i in a path P_k^t are present in WM and the $(r_j \in W,\ W \hookrightarrow r_i)$ fire, then r_i can fire.*

Proof When the $r_j \in W$ fire, WM contains at least the following facts:

$$\{\langle \lambda_i, l_i \rangle \mid \forall \langle \Lambda_i, L_i \rangle \in CT_{i,k}^t,\ (\exists \delta,\ \langle \Lambda_i, L_i \rangle \cdot \delta = \langle \lambda_i, l_i \rangle)\} \cup \{\langle \lambda_i, l_i \rangle \mid \forall r_j \in W,\ \langle \lambda_i, l_i \rangle \in \mathcal{A}^{r_j}\}$$

The templates that are matched by the facts in WM are given by

$$\begin{aligned} & CT_{i,k}^t \cup PT_{i,k}^t \\ = \ & (\mathcal{I}^{r_i} - PT_{i,k}^t) \cup PT_{i,k}^t \\ = \ & \mathcal{I}^{r_i} \end{aligned}$$

Thus, r_i can fire because WM contains facts matching all the templates in $\mathcal{I}^{r_i}$.

Now that we have understood the conditions required for an individual rule in a path to fire, we can specify the conditions for all the rules in a path to fire.

Theorem 4 *Given a path P_k^t, if facts matching the completion templates CT_k^t and start templates ST_k^t are present in WM, then all the rules in the path can fire.*

Proof The theorem follows directly from Lemma 2 by induction.

Thus, when assessing if all the rules in a path can fire, we must be able to determine if facts matching $ST_k^t \cup CT_k^t$ are present in WM.

The path model provides the CDPS designer with the ability to determine the data items that are required by a rule-based system to achieve its goals. In Theorem 1, we have shown that the precedence constraints for each path, ST_k^t and CT_k^t, indicate the facts required for all the rules in that path to fire, achieving a goal; the goal achieved is identified by the logical completion asserted by the rules in the path.

5.1.2 Determining Data Items Required: An Algorithm

Applying the path model to determine the facts required by the rules in each path to achieve a goal (ST_k^t and CT_k^t) required the addition of a new module to Path Hunter. The new module provides a list of the data items that are required to achieve each goal, SP_k^t and CP_k^t. The algorithm used to implement the module that we added to Path Hunter is shown in Figure 5.2. The first step in determining the facts required to achieve a goal is to compute the start templates ($ST_{i,k}^t$) and completion templates ($CT_{i,k}^t$) for each rule. The intuitive method for computing $ST_{i,k}^t$ and $CT_{i,k}^t$ would be to compute them for a rule, each time the rule appears in a path. However, we can show that $ST_{i,k}^t$ and $CT_{i,k}^t$ for rule r_i are independent of the path P_k^t in which r_i is contained.

Theorem 5 *When a rule is in two or more paths its set of completion templates is unique. Formally, if r_i is in path P_k^t and r_i is in P_n^t $(k \neq n)$, then $CT_{i,k}^t = CT_{i,n}^t$.*

Proof We will assume that r_i is in path P_k^t and P_n^t, but $CT_{i,k}^t \neq CT_{i,n}^t$. Since $CT_{i,k}^t \cup PT_{i,k}^t = CT_{i,n}^t \cup PT_{i,n}^t$, we note that $PT_{i,k}^t \neq PT_{i,n}^t$. That is, the predicates used in the facts asserted by W_1, $W_1 \hookrightarrow r_i$ for P_k^t are not the same as the predicates used in the facts asserted by W_2, $W_2 \hookrightarrow r_i$ for P_n^t. Therefore, $(\exists r_j \in W_1,\ \langle \lambda_i, l_i \rangle \in \mathcal{A}^{r_j},\ \langle \lambda_i, l_i \rangle = \delta \cdot \langle \Lambda_i, L_i \rangle,\ \langle \Lambda_i, L_i \rangle \in \mathcal{I}^{r_i})$, but $(\nexists r_l \in W_2,\ \langle \lambda_i, l_i \rangle \in \mathcal{A}^{r_l})$. This leads to the conclusion that W_2, is not maximal; thus, P_n^t is not a path. Since this contradicts our earlier assumption that P_n^t is a path, we conclude that $CT_{i,k}^t = CT_{i,n}^t$.

Corollary 4 *The facts required by a rule r_i in task T_t to fire that are not supplied by rules in T_t $(\{\langle \lambda_i, l_i \rangle \mid \langle \lambda_i, l_i \rangle = \delta \cdot \langle \Lambda_i, L_i \rangle,\ \langle \Lambda_i, L_i \rangle \in CT_{i,k}^t\})$ are independent of the*

```
get_data_items()                          {Determine data items required to achieve goals}
{
 Foreach r_i ∈ RB Do                      {Determine the facts required for a rule}
   If Start_Rule(r_i) then
     Find ST^t_{i,k}
   else
     Find CT^t_{i,k};
 Foreach path P^t_k ∈ RB Do               {Determine the facts required}
   Foreach r_i ∈ P^t_k Do
     If Start_Rule(r_i) then
       ST^t_k = ST^t_k ∪ ST^t_{i,k}
     else
       CT^t_k = CT^t_k ∪ CT^t_{i,k};
 Foreach path P^t_k ∈ RB Do               {Determine the type of data items required}
   SP^t_k = U(ST^t_k);
   CP^t_k = U(CT^t_k);
 Return()
}
```

Figure 5.2: Determining Data Items Required

path in which r_i is contained.

Thus, the facts that must be present in WM in order for a rule in a path to fire is independent of the path in which the rule is found.

Path Hunter will determine the data items required by a rule-based system to achieve goals, and the cost of determining data items required grows linearly as the the number of rules and paths in the rule base increase. In Theorem 2 and Corollary 2, we showed that we can calculate ST^t_k and CT^t_k without computing the facts required by a rule, for every path in which that rule appears; this reduces the computational complexity of determining data items required.

As a part of our experimental work, we used Path Hunter to analyze the paths in the rule base of the Blackbox Expert. The start predicates and completion predicates for the example path from chapter 4 are shown in Figure 5.3. The start predicates for the example path are $SP^{\texttt{Beam Analysis}}_{\texttt{example}}$ ={GMAP, GRIDSIZE, SHOT-RECORD}. The start predicates of our sample path indicate the data items that are required for the start rules in the path to fire, indicating that a specific configuration on the Blackbox grid is recognized. For the start rules in our example path, the Blackbox Expert requires access to data items that indicate the entry and exit points for the beams that have

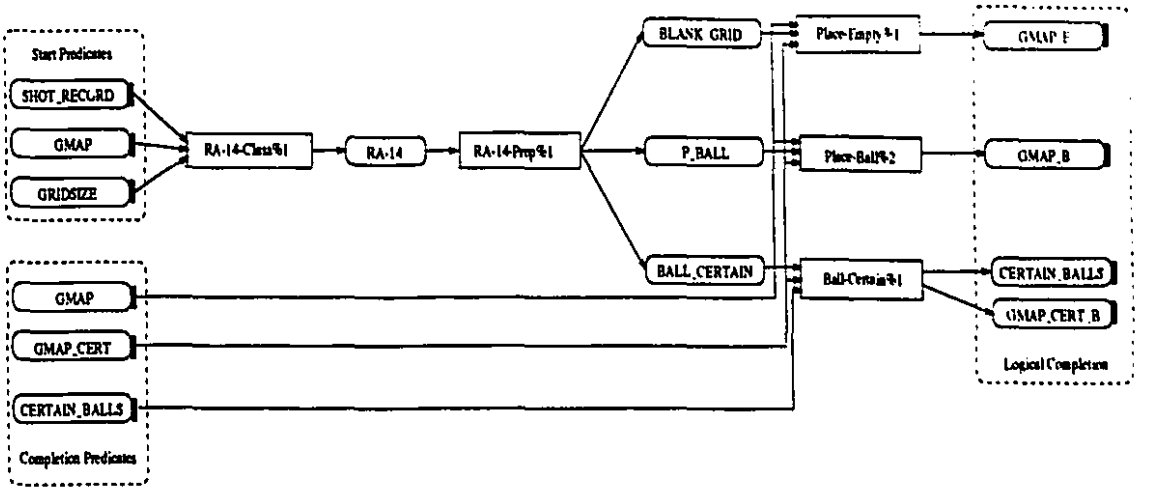


Figure 5.3: Completion and Start Predicates of a Path

been fired, the data item indicating the size of the grid used in the current puzzle, and data items indicating the current hypothesis for several of the Blackbox grid squares. The completion predicates for the example path $CP^{\texttt{Beam Analysis}}_{\texttt{example}}$ = {GMAP, GMAP_CERT, CERTAIN_BALLS}. The completion predicates in our sample path indicate the data items that are required for the non-start rules in the path to fire, indicating that a conclusion can be drawn as to the contents of several squares of the Blackbox grid. For the non-start rules in our example path, the Blackbox Expert requires access to data items that indicate the current hypothesis for several of the Blackbox grid squares, data items that indicate the certainty of hypotheses drawn for various squares of the Blackbox grid, and data items indicating the number of balls that have been located which are considered to be certain.

5.2 Monitoring Goals Achieved

Identifying each goal that was achieved by a rule-based system when it is problem-solving and the path that was responsible for achieving that goal requires a method to monitor the system at run-time. Our method to monitor rule-based systems relies upon the ability of the rule-based system to produce a *trace file* as it is problem-solving. The trace file contains the rule firing events which occurred when the system was exercised. In order to validate our method for detecting goals achieved, we show that Path Tracer (our tool for monitoring the goals achieved by a rule- based system)

is able to account for all the rule firing events that appeared in trace files obtained during the functional validation experiment we described in chapter 3.

5.2.1 Identifying Goals Achieved

Our method for monitoring a rule-based system to determine goals achieved is based upon detecting when all the rules in a path have fired; this indicates that the goal represented by the logical completion asserted by the rules in the path has been achieved, and indicates the path that is responsible for that goal having been achieved. Our method for identifying the goals achieved by a rule-based systems assumes that the rule-based system can produce a trace file. From this trace file, it must be possible to identify the sequence of rules which fired, and to identify the actual causal dependencies between the rule firings. We refer to run-time rule firings described in a trace file as *concrete* rule firings. A trace file describing this information can be obtained in several ways: many inference engines, such as the CLIPS inference engine, supply the required information; or it may be possible to modify an inference engine to produce the desired information, or one could instrument the rule base itself.

We faced several issues in applying our path model to determine the goals achieved and the path that was responsible for achieving each goal: identifying causal rule dependencies among the concrete rule firings, finding mappings from the concrete rule firings to abstract rules, and developing a method for determining when all of the rules in a path have fired. We will now discuss each of these issues in detail.

Finding Causal Run-time Dependencies: A trace file contains a linear sequence of rule firings. Within this sequence, causal sequences of rule firings are likely to be interleaved because of the opportunistic inference mechanism used by production rule systems. In order to determine true causal dependency relations, each asserted fact is assigned a unique identifier, so we can track which facts are "produced" and "consumed" by the rules. It is another useful feature of CLIPS that these identifiers are assigned automatically; otherwise, we would have had to instrument the rule base (instrumenting the rule base can be done automatically, of course). Given the existence of the unique fact identifiers, we are able to determine when a fact asserted by a concrete rule firing r_i^f helps to cause another concrete rule r_j^f to fire. If this is

the case, we say that r_i^f *causes* r_j^f *to fire*, indicated by the notation $r_i^f \gg r_j^f$.

Mapping From Concrete Rules to Abstract Rules: This is straightforward when the concrete rule has no conditionals on its RHS, because there will be only one corresponding abstract rule; otherwise, it is necessary to determine which one of several abstract rules corresponds to the rule which fired. In such cases, we are sometimes unable to make an unequivocal mapping from a concrete rule firing to an abstract rule because when a rule fires, not all of its expected assertions are observed. This is not a weakness of our model, since it points to one of two situations, each of which reveals information about the behavior of the rule base which is useful to the knowledge engineer:

- an assertion is not observed because (at least part of) the action performed by the rule has already been done by another rule, indicating that the rule firing is (at least partially) redundant; or
- the rule fails to assert some fact that it should assert, according to the specification of the rule-based system—this indicates a fault in the design of the rule and, since the rule violates the specification, of course there can be no corresponding abstract rule.

When a concrete firing cannot be mapped unequivocally to a single abstract rule, all the possible mappings are recorded as *equivocal mappings*. For example, if the sample rule, shown in Figure 4.1, `Ball-Certain` is observed (in the trace file) to fire at run-time, but it does not assert anything at that time because its first RHS conditional is false, then we cannot tell whether the concrete rule should correspond to `Ball-Certain%1`, or `Ball- Certain%2`. We allow it to correspond to *both* abstract rules, as an equivocal mapping. An unambiguous (unequivocal) mapping from the concrete rule firing r_i^f to the abstract rule r_j is denoted by $r_i^f \mapsto r_j$. An equivocal mapping from r_i^f to r_j is denoted by $r_i^f \overset{e}{\mapsto} r_j$.

Determining Rules Fired: Equivocal mappings complicate the task of determining when all the rules in a path have fired. One central question is: can we say that a specific abstract rule has been observed to fire when it participates in an equivocal mapping? We can argue that, on one hand, we do not want the abstract rule to be accepted as having fired unless we can be absolutely sure that it did so; on the other

hand, if the rule fired but failed to assert all the facts that it should, we can argue that the rule did fire. To capture this distinction, we use three different strategies for assessing if all the rules in a path have fired.

We illustrate our method for assessing if all the rules in a path have fired using an example; Figure 5.4(a) is a simplified representation for a path, showing its rules and their dependencies only. The path contains six dependencies, labeled $d_1, \ldots, d_6$. Assume that we are able to map concrete rule firings observed in the trace file to the abstract rules shown in Figure 5.4(a) as follows:

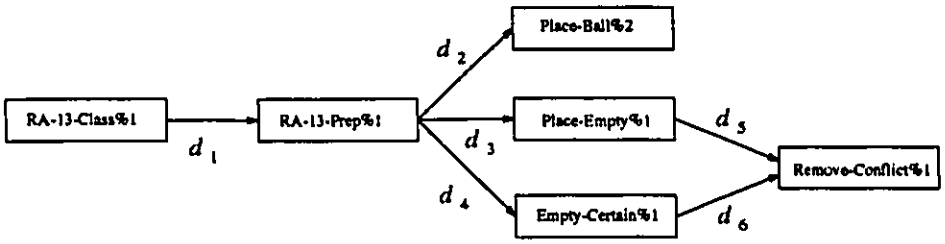
$$\begin{aligned}
\texttt{RA-13-Right}_i^f &\mapsto \texttt{RA-13-Class\%1} \\
\texttt{RA-13-Prep}_j^f &\mapsto \texttt{RA-13-Prep\%1} \\
\texttt{Place-Empty}f_k &\mapsto \texttt{Place-Empty\%1} \\
\texttt{Place-Ball}_l^f &\mapsto \texttt{Place-Ball\%2} \\
\texttt{Empty-Certain}_m^f &\mapsto \texttt{Empty-Certain\%1} \\
\texttt{Remove-Conflict}_n^f &\overset{c}{\mapsto} \texttt{Remove-Conflict\%1}
\end{aligned}$$

Assume also that we observe the following rule firing causalities in the trace file:

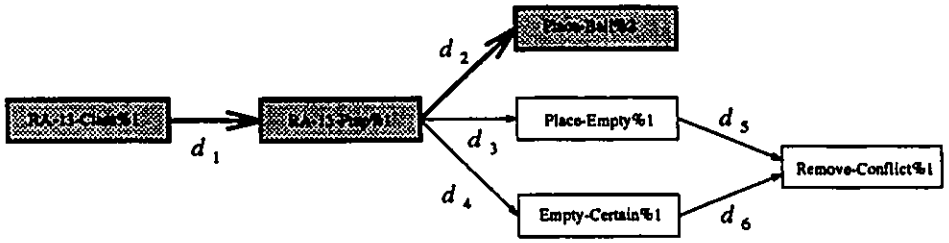
$$\begin{aligned}
\texttt{RA-13-Right}_i^f &\gg \texttt{RA-13-Prep}_j^f \\
\texttt{RA-13-Prep}_j^f &\gg \texttt{Place-Empty}_k^f \\
\texttt{RA-13-Prep}_j^f &\gg \texttt{Place-Ball}_l^f \\
\texttt{RA-13-Prep}_j^f &\gg \texttt{Empty-Certain}_m^f \\
\texttt{Place-Empty}_k^f &\gg \texttt{Remove-Conflict}_n^f \\
\texttt{Empty-Certain}_m^f &\gg \texttt{Remove-Conflict}_n^f
\end{aligned}$$

Paths are identified in a trace file using the notion of a *thread*, defined as follows: there is a thread $\mathcal{T}$ from rule r_1 to rule r_n in a path if the path contains the set of depends upon relations $\mathcal{T} = \{r_1 \prec r_2, r_2 \prec r_3, \ldots, r_{n-1} \prec r_n\}$. Sample threads in the example path are: $\{d_1, d_2\}, \{d_1, d_3, d_5\}, \{d_1, d_4, d_6\}$. A thread is said to be *observed* if and only if:

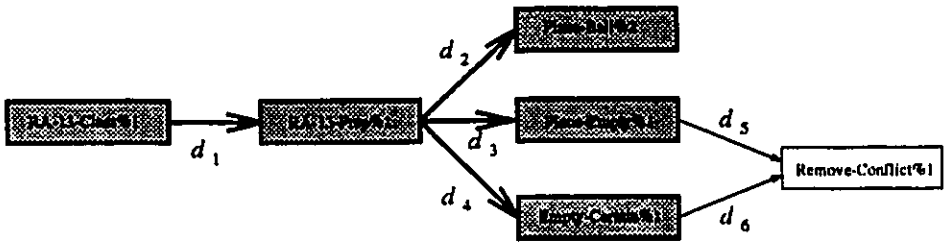
- there is a set of mappings $\{r_i^f \mapsto r_1, r_j^f \mapsto r_2, r_k^f \mapsto r_3, \ldots, r_l^f \mapsto r_{n-1}, r_m^f \mapsto r_n\}$, and



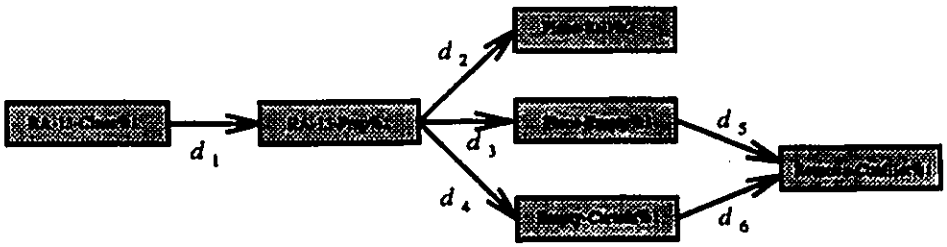
(a) An Example Path



(b) Rules Fired Using Conservative Strategy



(c) Rules Fired Using Moderate Strategy



(d) Rules Fired Using Liberal Strategy

Figure 5.4: Identifying Rules Fired in a Path

- there is a set of firing causalities observed in the trace $\{r_i^f \gg r_j^f, r_j^f \gg r_k^f, \ldots, r_l^f \gg r_m^f\}$.

Observed threads in the example path are: $\{d_1, d_2\}, \{d_1, d_3\}, \{d_1, d_4\}$. The thread $\{d_1, d_3, d_5\}$ is not observed because there is no unequivocal mapping to `Remove-Conflict%1` involving dependency d_5. Thread $\{d_1, d_4, d_6\}$ is not observed for the same reason.

We assess the rules that have fired in a path by examining dependencies between rules which are observed in the trace file. When we observe two rule firing events that are dependent upon each other, we can determine if these two rule firings can be safely accepted as belonging to a particular path. Dependent pairs of rule firing events are accepted based upon the position in a path of the dependency between the rules, and the position of any equivocal mappings in that path. Each strategy (conservative, moderate, and liberal) considers equivocal mappings differently for the purposes of accepting rule firings. The rules for deciding whether or not to accept a specific pair of dependent rules as being part of a path for each strategy are stated below.

Conservative Accept the dependent pair $r_i \prec r_j$ if and only if there is an observed thread $\mathcal{T}$ from r_s to r_e, where r_s is a start rule for the path, r_e is an end rule for the path, and $(r_i \prec r_j) \in \mathcal{T}$. Intuitively, we accept a thread from a start rule to an end rule in its entirety if every dependent pair in the thread is observed, and none of the rule mappings involved are equivocal; otherwise, we do not accept any dependent pairs in that thread as having been observed. Using this strategy, only the two pair of dependent rule firings comprising the example thread (d_1, d_2) are accepted, because all other threads from a start rule to an end rule involve an equivocal mapping. This is illustrated in Figure 5.4(b).

Moderate Accept the dependent pair $r_i \prec r_j$ if and only if there is an observed thread $\mathcal{T}$ from r_s to r_j, where r_s is a start rule for the path, and $(r_i \prec r_j) \in \mathcal{T}$. Intuitively, starting from the start rule of each thread, we accept observed dependent pairs until the first dependent pair involving an equivocal mapping, or until the end of the thread. We do not accept the actual dependent pair involving equivocal mappings. Using this strategy, four of the dependent rule pairs in the example (d_1, d_2, d_3, d_4) are accepted. This is illustrated in Figure 5.4(c).

Liberal Accept the dependent pair $r_i \prec r_j$ if and only if there is an observed thread from r_s to r_i, where r_s is a start rule for the path, $r_m^f \mapsto r_i$, $r_m^f \gg r_n^f$, and there is an equivocal mapping $r_n^f \overset{e}{\mapsto} r_j$. Intuitively, starting from the start rule of

each thread, we accept observed dependent pairs until the first dependent pair involving an equivocal mapping, or until the end of the thread, including the first dependent pair involving an equivocal mapping. Using this strategy, all six of the dependent rule pairs in the example are accepted, as is illustrated in Figure 5.4(d).

5.2.2 Validating Our Method

Before applying the path model to the design of any CDPS, it is important that we establish the validity of using the path model to determine if a rule based system has achieved a goal. In order to validate the accuracy of our method for detecting goals achieved, we use Path Tracer to analyze the rule firing events that appear in the trace files obtained during the functional validation experiment we described in chapter 3. Path Tracer provides two different outputs:

- for each path that Path Hunter has discovered in a rule base, Path Tracer produces a count of the number of rules in that path that have been accepted as having fired.
- Path Tracer produces a list of all rule firing events that it cannot accept as belonging to any path.

We will first examine a summary of the number of rules in each path that Path Tracer has accepted as having fired, and then show that we can account for all rule firing events listed by Path Tracer as not belonging to any path.

A summary of the results of using Path Tracer to analyze the trace files produced by the Blackbox Expert is given in Table 5.1. The number of rules fired is expressed as a percentage of the number of rules in a path; thus, the column marked 100% refers to paths in which all the rules fired. The entries in the table are expressed as a percentage of the paths in the rule base; thus, the entry 17.4 in the upper right-hand corner of the table indicates that all (100%) of the rules fired in 17.4% of the paths in the rule base when assessing rules fired using the conservative strategy. Using the liberal strategy, all rules fired in 33.3% of the paths. As we would expect, the number of paths in which all rules have been accepted as firing increases as the strategy becomes more generous (conservative to liberal). Note that the same number of paths were observed at $> 0\%$ rules fired using the moderate and liberal strategies,

Strategy	Percent Rules Fired						
	> 0%	> 50%	> 60%	> 70%	> 80%	> 90%	100%
Conservative	64.2	32.4	25.0	20.0	18.6	17.6	17.4
Moderate	71.7	50.6	44.6	36.5	27.5	19.0	18.4
Liberal	71.7	52.9	49.6	45.1	42.2	35.0	33.3

Table 5.1: Percentages of Rules Fired

indicating that no equivocal mappings were involved in any dependency which also involved a start rule.

The results produced by Path Tracer indicate that many of the paths that were discovered by Path Hunter are never exercised when the Blackbox Expert solved the test cases. This does not point to a problem with our method for determining the paths in which all rules have fired, rather Path Tracer has uncovered an incompleteness in our test set. This is not surprising since the test set was generated using only functional criteria. Most of the cases in which paths were never initiated at run-time are explained by the fact that the scenarios which those paths were designed to handle were not present in the test set. Therefore, the start rules for these paths never had cause to fire.

When Path Tracer analyzed the trace files produced by the Blackbox Expert, 6% of the total rule firings contained in the trace files could not be accepted as being part of any path. Path Tracer has rules to classify such firings, as follows:

Redundant Firings When a rule r_i fires and the facts it is to assert are already present in WM, due to actions taken by other rules, r_i does not assert any new facts, and cannot cause any other rule to fire; thus, r_i cannot participate in any mapping that indicates a depends upon relation, unless r_i is an end rule. If r_i is not an end rule, Path Tracer reports such concrete firings as *redundant firings*. The presence of a large number of these redundant firings that involve the same rule at run-time would be anomalous, and would suggest that there is some fault in the rule base.

Equivocal Firings Consider the example path shown in Figure 5.5. Let us assume that rules r_1, r_2, r_4, r_5 and r_7 are mapped unequivocally, but r_3 and r_6 are mapped only equivocally. Using the liberal strategy, the dependent pair $r_3 \prec r_6$ is not accepted because there is no observed thread from the start rule, r_1, to r_3 (the mapping to r_3 is equivocal). Since the liberal strategy is the most general of the three, dependent pair $r_3 \prec r_6$ is not accepted in the other strategies either. Therefore, the concrete firing which maps equivocally to r_6 is never ac-

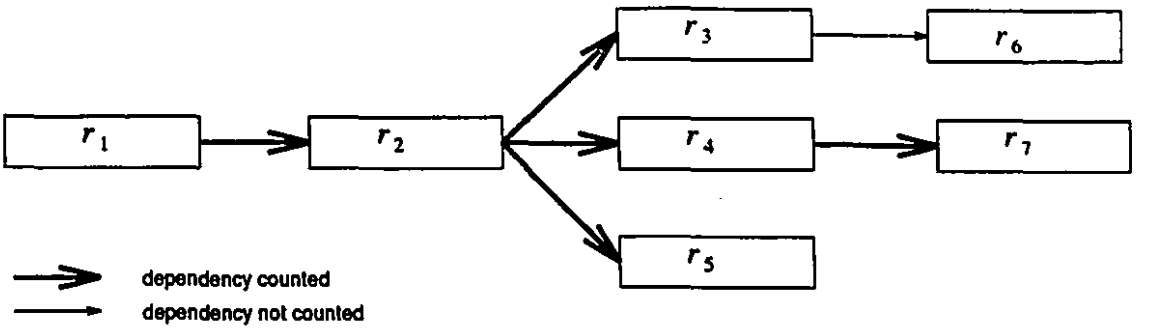


Figure 5.5: Rules Fired Using the Liberal Strategy

cepted as belonging to any abstract path by Path Tracer. Path Tracer reports such firings as *equivocal firings*. They are important because they indicate rule firings (usually near the end of paths) which fail to assert all of their expected predicates (hence the equivocal mappings). In many of the cases we have manually checked so far, this is because some other rule has "interfered" to carry out those assertions prior to the equivocal firing rules. Other cases pointed to coding errors in the rules, such as where a rule does not assert all the facts that it should.

All of the rule firing events in the trace files produced by the Blackbox Expert that could not be accepted as being part of any path were classified as belonging to one of the above cases: redundant firings or equivocal firings. This statement is highly significant, since it means that:

- the path model for rule base execution paths was sufficiently powerful to explain all the rule firing events observed in the trace files.
- the 516 paths produced by Path Hunter accurately described all the rule firing events observed during the functional testing of Blackbox Expert.
- the mechanisms used by Path Tracer for identifying causal concrete rule relationships, mapping from concrete rule firings to abstract rules, and identifying rules fired in a path, were capable of accounting for all of the rule firing events observed in terms of the paths discovered by Path Hunter in the Blackbox Expert's rule base.

These results provide us with strong empirical evidence for the validity of our structural model and tools. We can be confident in applying our model by using Path Hunter to discover the paths in a rule base, and by using Path Tracer to determine when all the rules in a path have fired; determining paths in which all rules have fired identifies each goal that is achieved by a rule-based system, and identifies the path which is responsible for achieving that goal.

5.3 Applying Our Models: A Case Study

We now apply our models to the design of a CDPS for solving the Blackbox puzzle. Using our organizational model, we discuss the inter-agent perspective: describing an organization for the CDPS in which the agents are rule-based systems constructed using the Blackbox Expert's rule base. Then we show the application of the path model to study the intra-agent perspective, considering the goals achieved by the Blackbox Expert as we reduced the data items available. We then demonstrate by means of three examples how our study of the intra-agent perspective aids the CDPS designer in setting data distribution.

5.3.1 The Inter-Agent Perspective

The specification of an organization for a CDPS to solve the Blackbox puzzle requires the CDPS designer to determine the following:

- the number and structure of the agents in the CDPS
- the coordination structure to be used by the agents
- the structure of the blackboard
- the window of each agent

For the purposes of our case study, we will assume that the CDPS contains three agents, and each agent contains the complete set of rules in the Blackbox Expert's rule base. As shown in Figure 5.6, the agents use Consensus to determine the beams to be fired [29].

The are many different options for representing the Blackbox grid using a blackboard; the CDPS designer must select the number of levels that are required, and the data items to be stored on each level. In the case of Blackbox, which is an ill-structured problem, the structure imposed by the problem-solver (the factors a_i that are used to represent the state of the Blackbox puzzle) will determine the number of levels required on the blackboard as well as the data items to be placed on the blackboard. More specifically, the "relationship" between the different objects to be placed on the blackboard will impact on the number of levels required. By relationship between objects on the blackboard, we refer to the manner in which the problem

solver will make use of the objects on one level of the blackboard to construct objects to be placed on the upper levels of the blackboard. For example, in the case of Blackbox, shot records placed on the lowest level of the blackboard are used as evidence to support hypotheses for the trajectory of the beams, and these hypotheses are recorded on higher levels of the blackboard than the shot records [68].

Figure 5.6 depicts the topmost level of the blackboard where the Blackbox gird is represented as a matrix, and the lowest level of the blackboard containing the beam entry and exit points are shown surrounding the grid. There would also be several intermediate levels (not shown in Figure 5.6) containing hypotheses as to trajectory of the beams that have been fired. On the lower levels of the blackboard, the hypothesis for beam trajectories would explain only a portion of the entire trajectory of a beam. On the upper levels of the blackboard, hypotheses from the lower levels on the blackboard would be used to construct hypotheses for the entire trajectory of a beam, and the hypotheses for entire beam trajectories would then be used to support the hypotheses for the contents of the grid squares, on the highest level of the blackboard [68].

In the absence of any input from examining the intra-agent perspective, at this stage we can consider only simple intuitive options for setting the window of each agent in the CDPS. Figure 5.6 shows the upper level of the blackboard divided into three approximately equal regions, and each agent can access only one of the regions. In the case of shot records (the lowest level on the blackboard), the agents can access a shot record if either the beam's entry or exit point is in the region of the blackboard it is allowed to access.

5.3.2 The Intra-Agent Perspective

Our study of the intra-agent perspective considers the ability of the Blackbox Expert to produce specific results, as the data items available are reduced. Using Path Tracer, we monitor the goals that are achieved by the Blackbox Expert as it solves a set of test puzzles with a reduced number of data items available. Analyzing the paths that are responsible for the achievement of these goals and the data items required for all the rules in these paths to fire, we determine the following:

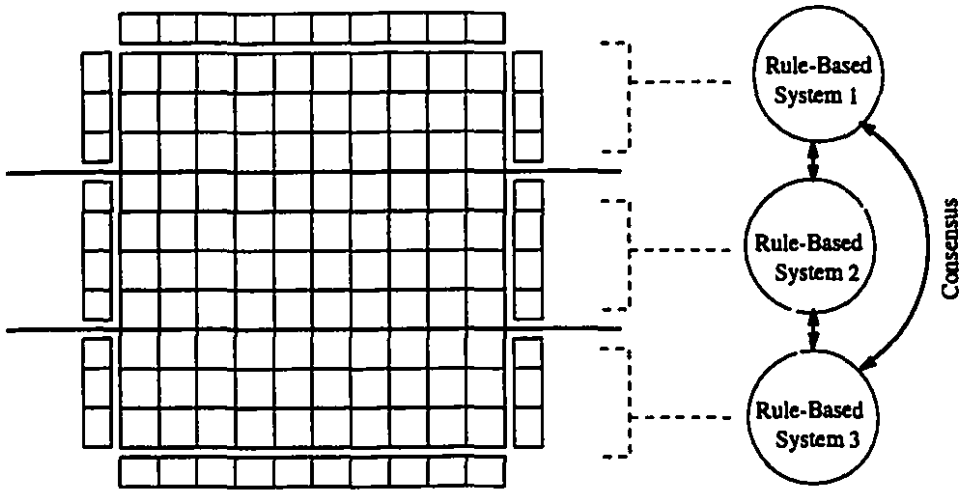
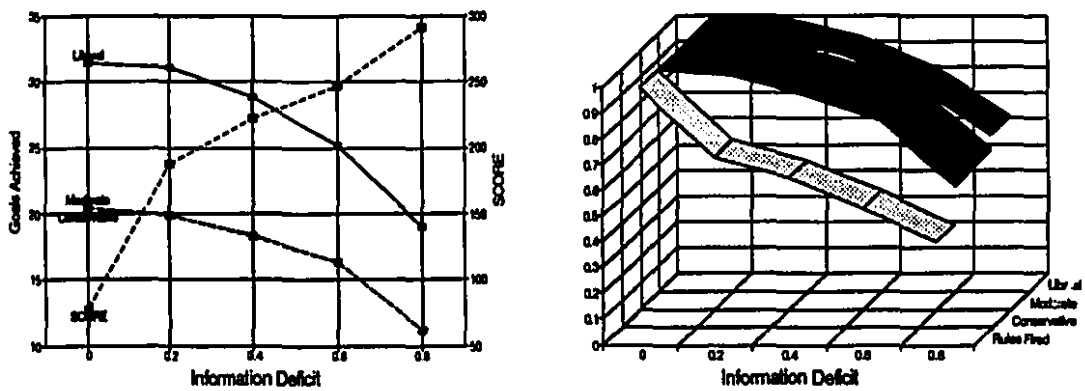


Figure 5.6: CDPS for Blackbox



(a) (b)

Figure 5.7: SCORE, Rules Fired, and Goals Achieved

- specific goals that are achieved by the Blackbox Expert and the result produced when those goals are achieved
- interactions that occur between goals
- the paths responsible for achieving goals that are identified by the CDPS designer as being undesirable; we also identify the data items required by all the rules in these paths to fire.

In our study, we selected 10 random test cases of the Blackbox puzzle, and the Blackbox Expert attempted each test case five times; each time the same test case was attempted, the availability of data items (measured using the information deficit metric) to the Blackbox Expert was reduced by 0.2. In order to ensure that our

experiment was unbiased, we used random test cases and we reduced the availability of each type of data item in equal proportions. We then used Path Tracer to analyze the trace files produced by the Blackbox Expert to determine the effect that the change in data items available to the Blackbox Expert had on the goals that were achieved. We used the *SCORE* metric (described in chapter 3) to measure the quality of the results produced by the Blackbox Expert for the test cases.

The overall effect of changing the data items available to the Blackbox Expert on the number of goals that were achieved is shown in Figure 5.7(a). This figure shows the number of goals that were achieved by the Blackbox Expert when solving our test set, as the data items available were reduced. Figure 5.7(a) also shows the average *SCORE* of the results produced by the Blackbox Expert. The SCORE of the results produced by the Blackbox Expert increased as the data items available decreased, and the number of goals achieved under each strategy decreased. Based upon our experiment described in chapter 3, the trends observed in these results are as expected.

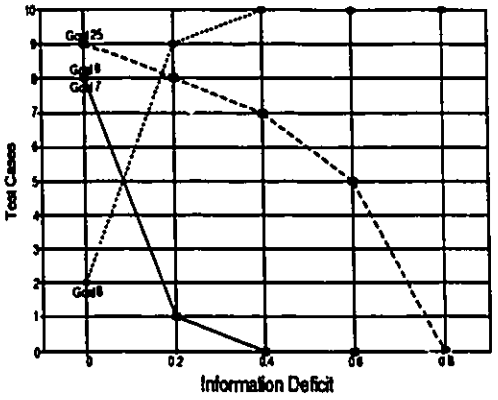
While the overall effects shown in Figure 5.7(a) are as one would expect, if we compare the rate of decrease in goals achieved with the rate of decrease for rules fired, we find an interesting anomaly. Figure 5.7(b) presents the ratio of rules fired at each information deficit to the number of rules fired with an information deficit of zero, and the ratio of the number of goals achieved at each information deficit to the number of goals achieved with an information deficit of zero. At information deficit 0.8, the number of rules fired decreases by 60%, but the number of goals achieved declines by only 39% (using the liberal strategy); even though the Blackbox Expert is still able to achieve over 60% of the goals it achieved with an information deficit of zero, it is nearly unable to solve any portion of the Blackbox puzzle (SCORE is 291).

Let us now consider the effect of reducing the data items available to the Blackbox Expert on its ability to achieve specific goals as well as the impact of achieving these goals on the ability of the system to produce a result, or the quality of the result produced. Figure 5.8(a) shows a sample of goals that were affected by the change in data items available to the the Blackbox Expert. For each goal, we plot the number of test cases in which the goals were achieved as the data items available were reduced.

Figure 5.8(b) explains the meaning attached by the knowledge engineer to each goal. Examining the paths that can achieve each goal, and the data items required by the rules in each path, the trends shown in Figure 5.8(a) can be interpreted as follows:

- goal 7 exhibits a rapid decline; goal 7 can only be achieved if sufficient data items are available for the Blackbox Expert to detect that all the balls in the grid have been located. This is only likely to occur in situations where most of the data items regarding the contents of the grid squares are available. The non achievement of goal 7 impacts the ability of the Blackbox Expert to detect when an end game situation has been reached. In the case that the Blackbox Expert is unable to detect end game situations, it tends to fire too many beams, resulting in an elevated score.
- goal 8 exhibits a rapid increase; goal 8 is achieved when the Blackbox Expert is unable to choose a beam to fire. As the number of data items used to evaluate the beams to be fired is reduced, the Blackbox Expert is unable to select a beam to be fired; thus, goal 8 is achieved more often as the number of data items is reduced. The achievement of goal 8 indicates that the reduced ability of the Blackbox Expert to select beams has been detected, and the Blackbox Expert should terminate its problem-solving efforts.
- goal 9 exhibits a rapid decline; as with goal 7, goal 9 can only be achieved if it is possible to detect that all the balls in the grid have been located. The non-achievement of goal 9 indicates the reduced ability of the Blackbox Expert to detect that it should terminate its problem-solving effort because all the balls have been located.
- goal 25 exhibits a decline, but not as rapid as the decline experienced by goal 7; The achievement of goal 25 is dependent upon data items that indicate the contents of different grid squares, that indicate the certainty of grid squares, and shot records for the beams that have been fired. As the data items available are reduced, it is less likely that the Blackbox Expert can detect the scenario required to achieve goal 25. The non-achievement of goal 25 indicates that the quality of the result produced by the Blackbox Expert is reduced because the Blackbox Expert is unable to determine the contents of the grid squares.

It is evident that when determining the impact of the availability of data items on the result produced by a rule-based system, not all goals can be considered to be equal in the role they play in the problem-solving process, in the effect they will have on the result produced by the rule-based system, or in sensitivity to a change in the availability of data items. Certain goals, while central to the problem-solving process (such as goals 7, 8, and 9 for the Blackbox Expert), may have little direct



(a)

GOAL	SEMANTICS
Goal 7	Detect if all balls located
Goal 8	End of game, no desirable beams
Goal 9	End of game, all balls located
Goal 25	Mark square as empty, and it is certain

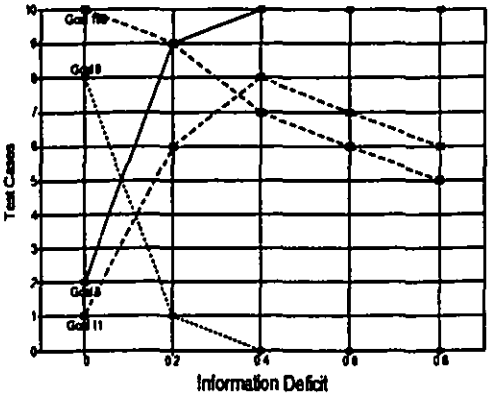
(b)

Figure 5.8: Sensitivity of Specific Goals

effect on the final result that is produced by the rule-based system, even if they can be achieved more often. Other goals, such as goal 25 for the Blackbox Expert, have a direct impact on the result produced each time they are achieved.

Let us now consider the interactions that occur between goals as the data items available to the Blackbox Expert are reduced. Figure 5.9(a) shows a sample of goals that interacted with each other as we changed the data items available to the the Blackbox Expert. For each goal, we plot the number of test cases in which the goals were achieved as the data items available were reduced, and Figure 5.9(b) explains the meaning attached by the knowledge engineer to each goal. Examining the paths that can achieve each goal, and the data items required by the rules in each path, the trends shown in Figure 5.9(a) can be interpreted as follows:

- goals 8 and 9 interact with each other. Goals 8 and 9 form a set of goals in which at least one of the members of the set has to be accomplished; problem-solving terminates when either goal 8 or goal 9 is achieved. Of course, problem-solving must terminate, even when the number of data items available is small; thus in each test case, either Goal 8 or Goal 9 is achieved. As the data items available are reduced, the number of test cases which terminate by achieving goal 9 reduce, causing the number of test cases which terminate by achieving goal 8 to increase.
- goals 11 and 198 interact with each other. The paths achieving goals 11 and 198 have the same start rules, and thus the same start sets. However, the completion sets for these two paths are different. Goal 11 is achieved when the data items



(a)

GOAL	SEMANTICS
Goal 8	End of game, no desirable beams
Goal 9	End of game, all balls located
Goal 11	Terminate line of reasoning
Goal 198	Mark square as empty, and terminate line of reasoning

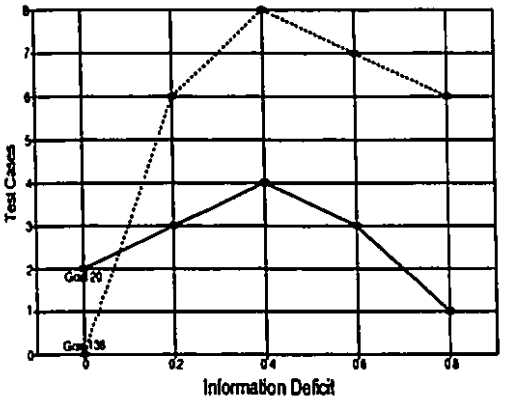
(b)

Figure 5.9: Interactions Between Goals

specified in the completion set of the path achieving goal 198 are unavailable, preventing the Blackbox Expert from achieving goal 198.

When the Blackbox Expert achieves goal 8 rather than goal 9, or goal 11 rather than goal 198 the quality of the result produced by the Blackbox Expert is reduced. In achieving goal 8 rather than goal 9, the Blackbox Expert would have fired more beams and identified the contents of fewer grid squares. In achieving goal 11 rather than goal 198 the Blackbox Expert is able to identify the contents of fewer grid squares. However, in each of theses case the Blackbox Expert is exhibiting a graceful degradation in its performance as the number of data items available are reduced; thus, even though the quality of the result produced by the Blackbox Expert is reduced when it chooses to achieve alternate goals as the number of data items available is reduced, we cannot categorize these goal interactions as undesirable.

We observed two goals that were achieved in a larger number test cases as the data items available were reduced, and we believe that an increase in the achievement of these goals is undesirable, see Figure 5.10. We observed an increase in the number of test cases in which goal 136 and goal 20 are achieved. The increased achievement of goal 20 indicates that the Blackbox Expert is placing additional balls as the number of data items available is reduced, and the increased achievement of goal 136 indicates that the number of conflicts encountered by the Blackbox Expert has also increased.



(a)

GOAL	SEMANTICS
Goal 20	Place a ball
Goal 136	Conflict on placing empty, and conflict on placing ball

(b)

Figure 5.10: Undesirable Events

Let us use the path model to look deeper into the cause for the two undesirable events that we observed in our experiment. Using Path Tracer, we were able to identify the paths that were responsible for the additional achievement of goal 136 and goal 20. In the case of goal 136, the paths we identified indicate that the Blackbox Expert produced the additional conflicts when it attempted to access data items for which it did not have access permission. We were able to identify two different sets of paths responsible for the achievement of goal 20 when the number of data items available were reduced. In one case, the Blackbox Expert erroneously placed a ball in a grid square (achieving goal 20). In the other case, the Blackbox Expert chose to fire a different beam than the beam it chose to fire with all data items available; firing a different beam produced a different set of data items, causing a ball to be placed in the Blackbox grid (achieving goal 20) rather than placing a ball that was also certain as well as marking several squares as empty.

Figure 5.11 shows the path responsible for achieving goal 136, and the path responsible for achieving goal 133; goal 133 was the goal achieved by the Blackbox Expert when all data items were available (the achievement of goal 133 indicates that the Blackbox Expert has successfully placed a ball, marked the ball as certain, and marked a square as being empty as well as certain). The increased achievement of goal 136 as the number of data items available to the Blackbox Expert are reduced occurs because of an interaction between goal 136 and goal 133. Both the paths

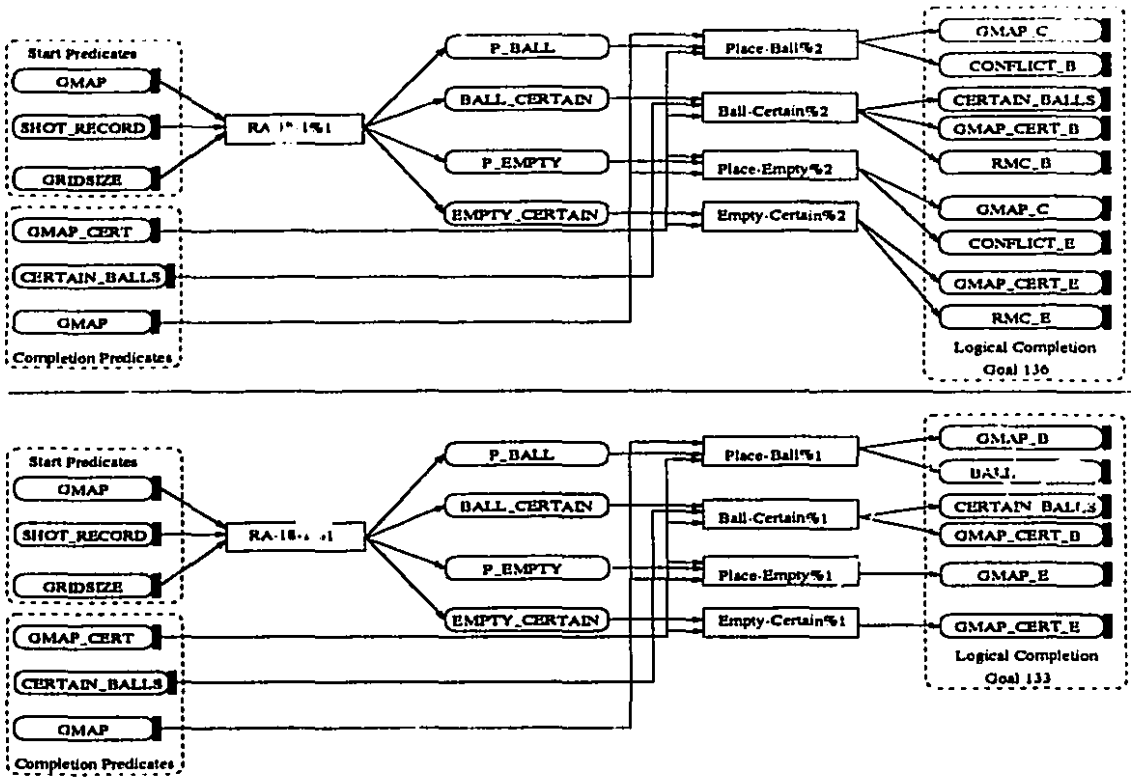


Figure 5.11: Paths Achieving Goal 136 and Goal 133

achieving goal 136 and goal 133 have the same start rule and require the same start predicates. As we reduced the data items available, the data items as identified by the start predicates for both paths were still available to the Blackbox Expert, but the data items identified by the completion predicates were not. As a result, all the rules in the path responsible for achieving goal 136 fire, but not all the rules in the path achieving goal 133 can fire; thus, goal 136 is achieved rather than goal 133, signaling a conflict. The conflict signaled by the Blackbox Expert indicates that although the Blackbox Expert has sufficient data items available to recognize a specific configuration in the Blackbox it is unable to draw a conclusion (updating grid squares) based upon that configuration because it required access to data items that were unavailable. This understanding of a conflict identifies an additional cause for conflicts, originally discussed in chapter 3, to include the situation where the Blackbox Expert attempts to update a grid square based upon a configuration that it recognizes, but it cannot due to access restrictions.

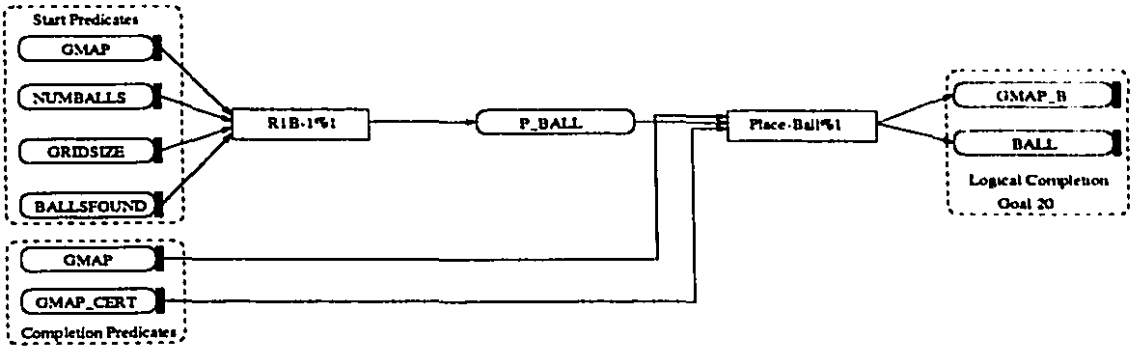


Figure 5.12: Path Achieving Goal 20

Figure 5.12 shows a path that was responsible for increased achievement of goal 20. In this case goal 20 is achieved, but it should not be achieved; thus, a grid square is marked as containing a ball by the Blackbox Expert, when in fact the square is empty. The erroneous achievement of goal 20 is due the the unavailability of the data items required by the Blackbox Expert to determine the number of grid squares that have not yet been identified. Rule `R1B-1%1` is designed to detect if there is only one grid square that is still unidentified, and only one ball not yet located. We observed that with a reduced number of data items available all the rules in this path fired, erroneously concluding that one of several remaining unidentified grid squares contains the one remaining ball. Of course, this is incorrect.

Figure 5.13 shows two sets of paths. The rules in each set of paths are activated depending upon the beam chosen to be fired by the Blackbox Expert. When all data items were available to the Blackbox Expert, the Blackbox Expert chose to fire a beam which lead to the firing of all the rules in the set of paths achieving goal 25, goal 88, and goal 147. The achieving of goal 25, goal 88, and goal 147 causes the Blackbox Expert to place a ball that is certain and mark several grid squares as empty, which are also certain. When the number of data items available were reduced, the Blackbox Expert chose to fire a beam which caused goal 20 to be achieved; achieving of goal 20, causes the Blackbox Expert to correctly place a ball, but the ball is not identified as being certain. The beam chosen with a reduced number of data items available causes the Blackbox Expert to produce an inferior result.

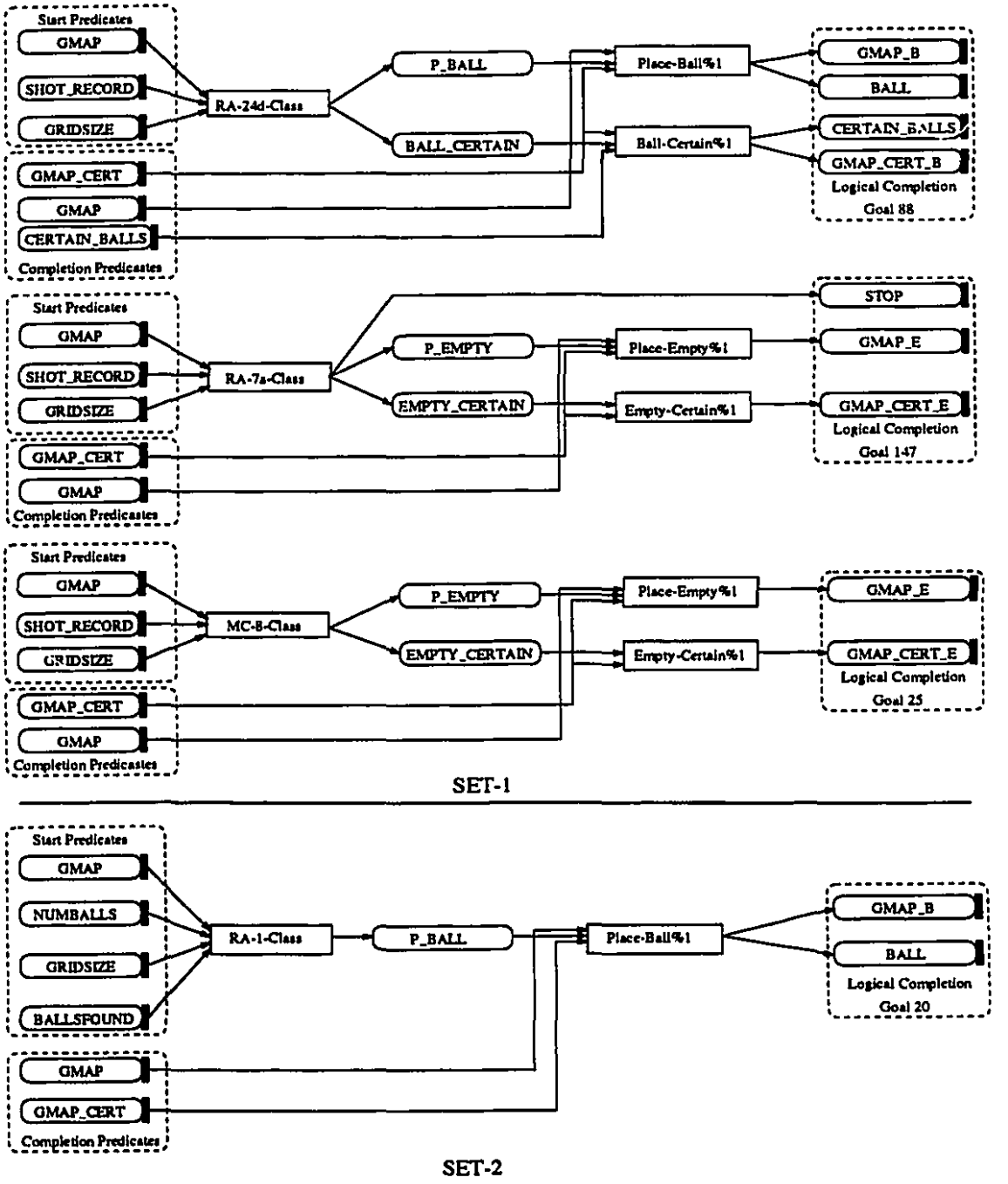


Figure 5.13: Two Sets of Paths

5.3.3 The Inter-Agent Perspective Revisited

Having studied the intra-agent perspective for the Blackbox Expert, we now reconsider the inter-agent perspective for the CDPS to solve the Blackbox puzzle, described in section 5.3.1. We will focus on demonstrating how the "information" gained by studying the intra-agent perspective can be used by the CDPS designer in setting data distribution, in order to improve performance. In studying the intra-agent perspective (see section 5.3.2), we observed an increase in the achievement of goal 20 and goal 136, and we determined that this increase was undesirable because the performance of the Blackbox Expert was adversely affected. Using Path Tracer, we were able to isolate the paths that were responsible for the increased achievement of these goals; and using Path Hunter, we were able to determine the data items required by all the rules in each path to fire.

As a demonstration of the use of the information gained from studying the intra-agent perspective, we present three scenarios in which the CDPS designer can avoid an increase in the achievement of goal 20 or goal 136, by using the intra-agent perspective as a guide in adjusting the data distribution in the CDPS for solving Blackbox. In developing these scenarios, we used Path Tracer to identify the specific test case in which the increased achievement of goal 20 and goal 136 occurred, indicating the data distribution that was in effect when the Blackbox Expert solved these test cases. We developed each scenario by examining the paths responsible for achievement of goal 20 and goal 136, the data items required by all the rules in the paths responsible for achieving these goals to fire, and the data distribution that was in effect when there was an increase in the achievement of these goals; giving us the conditions in which the increased achievement of these goals occurred. We then determined if it would be possible for the conditions responsible for the increased achievement of goal 20 or goal 136 to be replicated in the CDPS for solving Blackbox.

We developed one scenario for each of the paths, or sets of paths, that were identified as being responsible for the increased achievement of goal 20 or goal 136 (see section 5.3.2). In each scenario, we first describe how the conditions responsible for the increased achievement of goal 20 or goal 136 can be replicated in the CDPS for solving Blackbox, and then we discuss how the data distribution in the CDPS can

be modified to avoid the increased achievement of the goal. We are able to suggest a modification to the data distribution by examining the path that was responsible for the increased achievement of the goal, and examining the data items required for all the rules in that path to fire.

Scenario 1: Let us now consider how the conditions that lead to an increase in the achievement of goal 136 can occur in the CDPS for solving Blackbox. The CDPS shown in Figure 5.14 depicts the situation where a beam has been fired, and its entry point and exit point are labeled by 'A'. Rule-based system 3 has access to the data items indicating the entry and exit point of the beam. The data items available within the window of rule-based system 3 are sufficient for the start rule (RA-18-1%1) of both paths shown in Figure 5.11 to fire. Given the access privileges for rule-based system 3 as shown in Figure 5.14, goal 136 will be achieved because rule-based system 3 does not have access to the grid squares labeled by 'C'. The conditions that lead to an increase in the achievement of goal 136 are that rule-based system 3 has sufficient data items available in its local view to recognize a specific configuration on the Blackbox grid, but it does not have access to grid squares labeled by 'C' which it requires in order to draw a conclusion based upon the configuration it has recognized.

The conditions that lead to an increase in the achievement of goal 136 can be avoided by adjusting the data distribution in the CDPS. Clearly, the original specification for the window of each rule-based system is inadequate. Rule-based system 3 is able draw a conclusion based upon the data items available within its own window which impacts on rule-based system 1's and on rule-based system 2's window. The data distribution in the CDPS should be adjusted to permit rule-based system 3 to access the portions of the Blackbox grid for which it can draw conclusions given the data items available for recognizing specific configurations in the Blackbox, rather than simply concluding that a conflict has occurred.

This scenario is an example of how applying our models to study CDPS, we can determine that the setting for the data distribution in the CDPS is inadequate, because a rule-based system in the CDPS has sufficient data items available to be able to draw a conclusion, but it does not have access to the region that is impacted by the conclusion. Given the windows for each of the rule-based systems in the CDPS, we

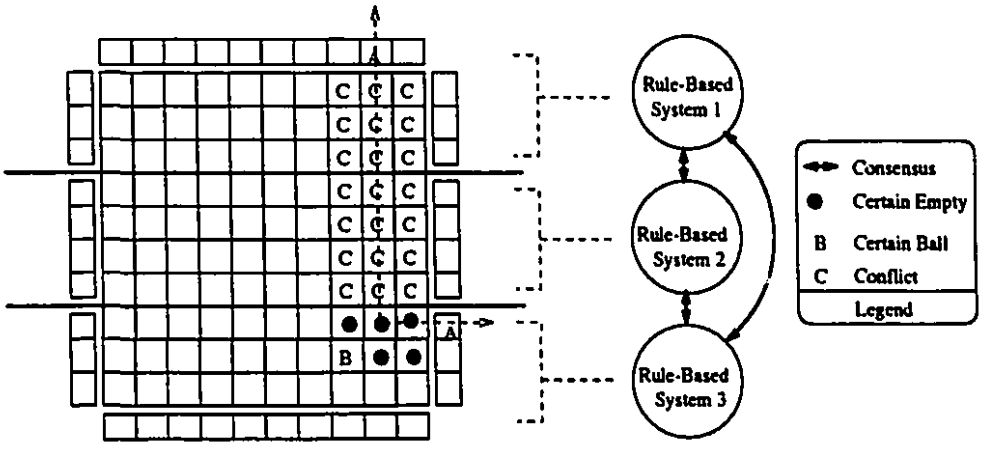


Figure 5.14: Conclusion Draw Outside Window

determined that one of the rule-based systems is able to draw a conclusion based upon the data items available with its local view. However, the conclusion drawn by the rule-based system impacts the local view of another rule-based system in the CDPS. Typically, this situation would be avoided by adjusting the windows of the rule-based systems in the CDPS to permit the rule-based system drawing the conclusion to share the data items it is able to produce.

Scenario 2: Let us now consider how the conditions that lead to an increase in the achievement of goal 20, by erroneously placing a ball, can occur in the CDPS for solving Blackbox. An example of how the path we identified as being responsible for the achievement of goal 20 can cause a rule-based system in our CDPS to erroneously place a ball is shown in Figure 5.15. There is one ball that has not yet been located in rule-based systems 1's window. Rule-based system 3's local view, given by its window, shows that five of the six balls have been located and only one unmarked grid square remains. All the rules in the path shown in Figure 5.12 fire, and goal 20 is achieved, concluding that the unidentified grid square must contain the one remaining ball; thus, rule-based system 3 incorrectly places a ball in its own window, when in fact the ball is located in rule-based system 1's region. The conditions that lead to an increase in the achievement of goal 20 are that rule-based system 3 is drawing a conclusion based upon the data items available in its window, but the data items indicating the number of unidentified grid squares must reflect a global view.

The conditions that lead to an increase in the achievement of goal 20 by er-

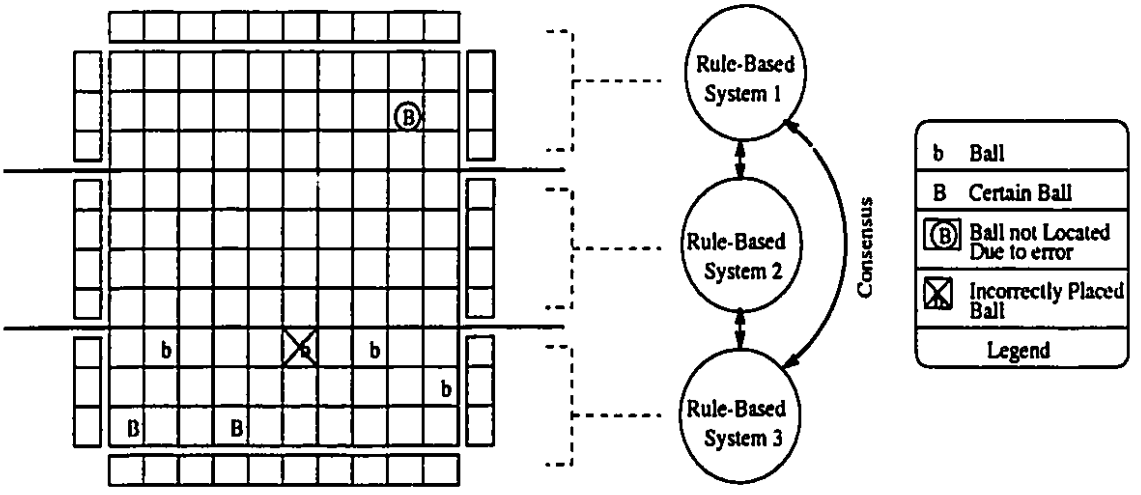


Figure 5.15: Local vs Global View

roneously placing a ball can be avoided by providing a mechanism by which the rule-based systems in the CDPS can determine the global number of unidentified grid squares. Our analysis of the path that is responsible for achieving goal 20 indicates that the conclusion drawn when all the rules in that path fire is dependent upon obtaining data items required to determine the total number of unidentified grid squares.

This scenario is an example of how applying our models to study CDPS, we can determine that the setting for the data distribution in the CDPS is inadequate, because a rule-based system in the CDPS is dependent upon the availability of data items whose value must reflect a global view of the problem-solving state.

Scenario 3: Let us now consider how the conditions that lead to an increase in the achievement of goal 20 due to a different beam being fired can occur in the CDPS for solving Blackbox. In Figure 5.16 we draw attention to the bottom row of the grid, which is in rule-based system 3's window. Several beams have been fired, and many of the grid squares have been identified as being empty and certain. In the case that a beam is to be fired into the grid from the right side, as indicated in Figure 5.16, all the rules in the set of paths achieving goal 88, goal 147, and goal 25 (shown in Figure 5.13) will fire, placing a ball that is certain and marking several grid squares as empty. In the case that a beam is fired into the grid from the bottom, all the rules in the path achieving goal 20 (shown in Figure 5.13) will fire, placing a ball that is not

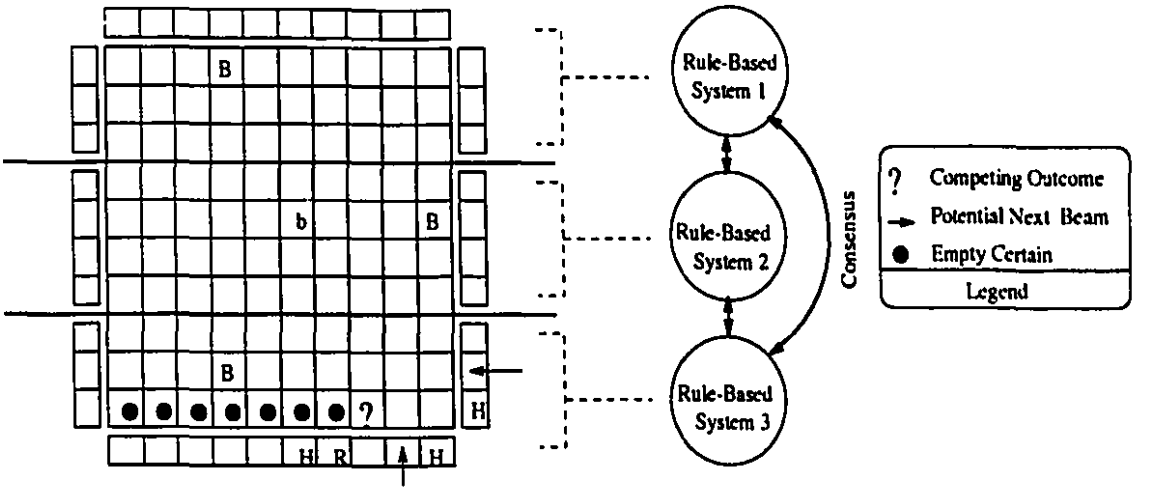


Figure 5.16: Selecting a Beam to Fire

certain. Given the local view of rule-based system 3, it would choose to fire the beam into the grid from the bottom. The conditions leading to this choice by rule-based system 3 are that rule-based system 3 is unaware of the balls that have been located in rule-based system 2's region.

The conditions that lead to an increase in the achievement of goal 20 can be avoided by ensuring that rule-based system 2 shares data items indicating the location of the balls located in its region with rule-based system 3 when selecting the next beam to fire. The path model identifies different sets of paths in which all rules will fire, given the beam that is chosen to be fired. In Consensus, rule-based system 2 can determine that it must supply data items indicating the position of the balls that have been located in its region, in order to indicate to rule-based system 3 that firing the beam from the right side is better than firing the beam from the bottom.

This scenario is an example of how applying our models to study CDPS, we can determine that the setting for the data distribution in the CDPS may be inadequate, because a rule-based system fails to share data items which are relevant to the options being considered by the rule-based systems during Consensus. When a rule-based system in a CDPS is able to determine the sets of paths in which all the rules in one of the sets can become enabled to fire, depending upon the option chosen by the agents in a CDPS.

5.3.4 Summary

We demonstrated the application of the organization and path models to the design of a CDPS for solving the Blackbox puzzle, focusing on setting data distribution. Using the organizational model, we considered the inter-agent perspective for the CDPS; and using the path model, we studied the intra-agent perspective. In our study of the intra-agent perspective, we identified three events in which a path, or a set of paths, were responsible for an increase in the achievement of goals, and the increased achievement in these goals was determined to be undesirable. Then we demonstrated, by presenting one scenario for each undesirable event, how a study of the intra-agent perspective can be used by the CDPS designer in adjusting the data distribution in a CDPS to improve performance as follows:

Scenario 1 the path model identified a path in the rule base of the Blackbox Expert which would produce superior results if specific data items were made available by adjusting the windows of the rule-based systems.

Scenario 2 the path model identified a path in the rule base of the Blackbox Expert which is dependent upon the availability of data items whose value must reflect a global view of the problem-solving state in order to avoid placing a ball in a grid square that is in fact empty.

Scenario 3 the path model identified different sets of paths in the rule base of the Blackbox Expert in which all the rules in one of the sets can become enabled to fire, depending upon the beam selected. These sets of paths indicate the data items that should be shared to choose the option that will produce the best result.

The precise steps we followed in applying our models to study the design of a CDPS for Blackbox were as follows:

The Intra-Agent Perspective

1. We used Path Hunter to determine the paths in the rule base of the Blackbox Expert as well as the data items required by the rules in each path to fire.
2. We used the Blackbox Expert to solve a set of test cases while reducing the number of data items available.
3. We used Path Tracer to analyze the goals achieved, the paths responsible for achieving these goals, and result produced by the Blackbox Expert as it was problem-solving with a reduced number of data items. The specific issues that were studied are as follows:

- the ability of the Blackbox Expert to achieve specific goals, and the impact of achieving those goals on the result produced.
- the interactions that occurred between goals.
- the data items required by the paths responsible for the achievement of the goals where the achievement of that goal was considered to be undesirable because its achievement adversely affected performance.
- for each path identified as being responsible for the achievement of a goal where the achievement of that goal was considered to be undesirable, we determined the data distribution that was in effect when all the rules in that path fired.

The Inter-Agent Perspective

1. We used the organizational model to specify a CDPS for solving the Blackbox puzzle.
2. For each goal that is adversely affected by a reduction in the availability of data items (identified by studying the intra-agent perspective), we determined the conditions which led to the goal being achieved.
3. We determined if it would be possible for the conditions responsible for the achievement goals that adversely affect performance can be replicated in the CDPS for solving Blackbox.
4. We determined how to adjust the data distribution in the CDPS to avoid the achievement of goals that adversely affect performance, by examining the path that was responsible for the increased achievement of the goal, and examining the data items required for all the rules in that path to fire.

5.4 Conclusion

We have applied our organizational and path models to building CDPS, by considering both the inter-agent perspective and the intra-agent perspective. In this thesis, we have discussed the use of our models only in aiding the CDPS designer in setting data distribution. Applying our models, the CDPS designer is able to determine how to set the availability of specific data items in order to improve the performance of the rule-based systems in a CDPS, or avoid serious failures. The organizational model permits the CDPS designer to specify interactions between the rule-based systems in a CDPS (the inter-agent perspective), and the path model permits the CDPS designer to analyze from the intra-agent perspective the effect of a reduction in the availability of data items as follows:

- determine the specific goals that are achieved and the result produced by achieving those goals; Also, identify the path that is responsible for the achievement of each goal.
- determine the interactions that can occur between goals.
- determine how to avoid the achievement of goals that are identified by the CDPS designer as being undesirable, by analyzing the paths responsible for the achievement of the undesirable goals, obtaining the data items responsible for all the rules in the path to fire.

Using the Blackbox puzzle as a sample ill-structured problem, we have demonstrated that a study of the intra-agent perspective provides guidance to the CDPS designer in improving the performance of the rule-based systems in a CDPS as follows:

- paths are identified in the rule base of an agent, which could produce superior results if specific data items were available, guiding the CDPS designer in setting the data distribution in the CDPS.
- paths are identified which are dependent upon the availability of data items whose value must reflect a global view of the problem-solving state in order to avoid producing erroneous results.
- different sets of paths are identified in which all the rules in one of the sets can become enabled to fire, depending upon the option chosen by the agents in a CDPS. Thus, the paths can be used to determine the data items that should be shared between the agents in a planning group, enabling a choice of the option that will produce the best result.

In applying the organizational and path models, we have used Blackbox as our sample problem, and the Blackbox Expert as our sample rule-based system. We believe that these models can be applied to other rule-based systems following the steps outlined in section 5.3.4. In order to use our models and tools in setting the data distribution in a CDPS for solving ill-structured problems in general, several assumptions must be satisfied:

1. there must exist an implementation of the agents to be used in the CDPS, and the agents must be implemented as rule-based systems.
2. the knowledge engineer must be able to identify the set of logical completions for the problem solved by the rule base.
3. there must exist a "suitable" set of test cases.
4. there must exist a method for limiting the data available to the rule-based system as it solves the test cases.

5. the rule-based system must be able to produce a trace file recording the rules fired, when it is exercised.
6. the CDPS designer must be able to identify goals that adversely affect the result produced when they are achieved.
7. the CDPS designer must be able to identify goals that must be achieved in order to produce an acceptable result.

One difficulty that will be faced by a CDPS designer in applying the path model to study the intra-agent perspective is to determine if a given a test set will ensure detection of all conditions under which a goal is achieved that adversely affects performance. In order to ensure that a study of the intra-agent perspective will identify all cases in which goals are achieved that adversely affect performance, the CDPS designer must determine if all paths in the rule base have been exercised when studying the intra-agent perspective; this is referred to as maximizing the *coverage* obtained on the rule base when the rule-based system is used to solve the test set.

Chapter 6

Conclusion

"The important thing is not to stop questioning."
- Albert Einstein

In this thesis, CDPS is viewed from two different perspectives: the inter-agent perspective, and the intra-agent perspective. The inter-agent perspective is concerned with issues relating to the manner in which the agents in a CDPS interact: the manner in which the problem is decomposed, the tasks assigned to each agent, the sharing of data items during planning and during execution, and the planning protocols used by the agents. The intra-agent perspective is concerned with the internal structure of the agent, and how the agent is affected by the environment in which it must operate — the CDPS. In addition to this dichotomy, in our research we have followed two complementary tracks: theoretical and experimental. We have developed formal models for both the inter-agent and the intra-agent perspectives (our models for the intra-agent perspective apply only to rule-based systems), developed an experimental testbed for an agent implemented as a rule-based system, and we have applied our models to aid the CDPS designer in setting the data distribution in a CDPS, demonstrating the benefits to the CDPS designer of using our models.

With the intent of studying the intra-agent perspective, we have developed a model for the structure of a rule-based system that models a rule base as a set of entities called "paths". Our model makes use of the dependencies between the rules in the rule base to define paths. Based upon our study of the problems with previous attempts by researchers to define models for the structure of a rule base, we believe that structural

models for rule based systems should satisfy three criteria: accuracy, meaningfulness, and computational tractability. Our structural model meets these three criteria: thus, our model is an improvement over previous attempts by researchers to capture dependencies between the rules in a rule-based system.

We apply the path model to study the intra-agent perspective by using paths to analyze the effect of a reduction in the availability of specific data items on the result produced by a rule-based system. The path model is applied in two separate steps: each path is used to identify the data items (start templates and completion templates) required by the rule-based system to achieve one of its goals (logical completion), and then paths are used to determine the goals that are achieved by a rule-based system as it solves a set of test cases; we also monitor the result produced for each test case. When the rule-based system is solving the test cases, we reduce the data items available; this permits us to determine interactions that occur between goals, and to determine the data items responsible for the achievement of specific goals whose achievement is considered to be undesirable by the CDPS designer.

We have constructed the Blackbox Expert, a testbed for experimental research. The Blackbox Expert is designed to permit us to perform experiments in which we change the data items available to a single rule-based system while we observe the functional and computational performance of the system, simulating the environment that the Blackbox Expert would experience as a member of a CDPS. We selected the Blackbox puzzle (the problem solved by the Blackbox Expert) as a sample ill-structured problem, which is suitable as a testbed application for CDPS: solving the Blackbox puzzle requires a rule-base that is sufficiently complex (containing several hundred rules) to serve as a realistic testbed, the effort required by a human to become proficient at solving the puzzle is small (less than one person-week), and the Blackbox puzzle can be used to model diagnosis type problems, a common application for rule-rule-based systems. Before using the Blackbox Expert in our research, we believed that the ability of the Blackbox Expert operating as a single agent solving the puzzle had to be established; thus, we validated the functional performance of the Blackbox Expert in solving Blackbox puzzles experimentally, establishing a baseline with which we could compare the performance of the Blackbox Expert as we changed the data

items available.

We have constructed a rule-base analysis tool called Path Hunter, which embodies our structural model. Path Hunter analyzes a rule base to determine the paths it contains, including the data items required by each path (start templates and completion templates) and the goal that will be achieved when all the rules in a path fire (logical completion). We have used Path Hunter to analyze the rule base of the Blackbox Expert, obtaining the set of paths in the Blackbox Expert's rule base. Using Path Hunter to analyze the structure of the Blackbox Expert's rule base has provided us with an opportunity to demonstrate that our structural model does in fact meet our three criteria. In order to use Path Hunter to analyze the structure of the Blackbox Expert's rule base, we had to specify all of the logical completions for the Blackbox problem; the logical completions are an input to Path Hunter.

We have developed a second tool, Path Tracer, embodying our method for applying the path model to monitor the goals achieved by a rule-based system. Path Tracer analyses trace files produced by a rule-based system when it is problem-solving; a trace file contains a list of all the rules fired by the rule-based system. Path Tracer will also identify the path that is responsible for achieving a goal, by determining the path in which all the rules have fired, asserting the logical completion representing that goal. In order to validate the accuracy of Path Tracer in monitoring goals achieved, we used Path Tracer to analyze the trace files produced when we validated the functional performance of the Blackbox Expert. Path Tracer is able to account for all the rule firing events that are recorded in these trace files, demonstrating the accuracy of the mechanism used by Path Tracer to identify the rules in a path that have fired.

We have developed a model for CDPS which allows for a precise description of the interactions that are permitted between the agents in a CDPS (the inter-agent perspective), providing a specification of the manner in which the agents in a CDPS are to cooperate. We refer to our model as an organization. Work by other researchers has referred indirectly to the types of interactions that could occur between the agents in a CDPS, but there had been no precise definition of the components of a CDPS and the types of interactions that should be specified when designing a

CDPS. Our organizational model also includes the information deficit metric, which is a metric introduced in this thesis for measuring the data distribution in a CDPS. The information deficit metric allows the CDPS designer to measure the availability of data items (data distribution) to the agents in a CDPS when they communicate with each other using a blackboard.

Our organizational model provides a basis upon which the design parameters for CDPS can be studied from an inter-agent perspective. We show the use of our organizational model and the Blackbox Expert to study the trends in performance that can be expected as the data distribution in a CDPS is modified. While many researchers have speculated as to the effect of the data distribution in a CDPS on the performance of the agents in that CDPS, we lacked evidence that the trends expected by researchers would actually occur when the agents in a CDPS were implemented as rule-based systems. Using the information deficit metric to measure the data items available to the Blackbox Expert as it solved a set of random test cases, we were able to statistically verify that the trends expected by researchers in functional and computational performance occurs for the Blackbox Expert solving any Blackbox puzzle as the availability of data items changed. Our experiment confirmed that the trends expected by researchers can actually occur in operational rule-based systems, and establishes that data distribution is an important factor in determining the performance of the rule-based systems in a CDPS. Thus, our study of global trends in performance confirms that the CDPS designer must be concerned with the data items available to each rule-based system in a CDPS when attempting to maximize performance.

We have demonstrated the usefulness of our models (paths and organizations) to the designer of CDPS, by considering both the inter-agent and the intra-agent perspective in the design of a CDPS for solving the Blackbox puzzle; we focused on applying our models to aid the CDPS designer in setting the data distribution in the CDPS. Using the organizational model, we presented a CDPS for solving Blackbox where the agents in the CDPS are constructed using the rule base of the Blackbox Expert. Then using the path model, we studied the ability of the Blackbox Expert to achieve its goals as the number of data items available (measured using the information deficit metric) was reduced. Using Path Hunter and Path Tracer, we determined

the result produced by the Blackbox Expert when a specific goal is achieved, determined the interactions that occurred between goals, and determined the goals whose achievement were adversely affected by the reduced number of data items available. We then identified the paths that were responsible for the achievement of adversely affected goals, allowing us to isolate the specific data items, or lack of data items, that caused the behavior we observed. We then presented three scenarios explaining how our study of the effect of data items available on goals achieved can be used to improve the performance of the rule-based systems in the CDPS, which are summarized as follows:

- we were able to identify data items which should be shared among the agents in the CDPS, indicating that an adjustment to the windows of the agents was required.
- we were able to identify data items whose value must reflect a global view of the state of the Blackbox problem in order to avoid an agent producing erroneous results.
- we were able to identify sets of paths that could be used to evaluate the potential outcome of selecting different beams to be fired, permitting the selection of the beam that is likely to produce the better result.

To summarize, we believe that our use of both formal models and experimental systems serves as an example of a research project in which theoretical results are applied to practical systems, validating the theoretical models. We applied our organizational model to the design of the Blackbox Expert, permitting us to implement a mechanism by which we can control the data items that are available to the Blackbox Expert when it is problem-solving. The information deficit metric has been applied in measuring the data items available to the Blackbox Expert when problem-solving, establishing statistically the existence of a relationship between the data available to a rule-based system and its performance. Our structural model has been embodied into the rule-base analysis tools, Path Hunter and Path Tracer. Path Hunter has been used to analyze the rule-base of the Blackbox Expert, validating the Path model. Path Tracer has been used to analyze the goals that the Blackbox Expert chooses to solve at run-time when the data items available to the Blackbox Expert are reduced. Using the data provided by our experiments, we have been able to provide examples and show how our models and tools can be used by the CDPS designer.

6.1 Future Work

In this thesis, we presented two models (the organizational model and the path model), two tools (Path Hunter and Path Tracer), and a testbed (the Blackbox Expert) for performing experiments examining the effect of data availability on the performance of a rule-based system. By considering the assumptions made in this thesis, or possible extensions to the line of research described in this thesis, we identify several problems for future research that we believe would be of interest:

- we believe that the experimental track followed in this thesis can be extended with the development of a multiagent version of the Blackbox Expert which solves a distributed version of the Blackbox puzzle
- the theoretical track can be extended to provide the CDPS designer with a formal method for using the information gained by studying the intra-agent perspective to set the data distribution in a CDPS.
- we also envisage the application of the path model to aid in the design of test sets in which the coverage of the rule base would be considered acceptable by the CDPS designer.

We will now discuss each of these three avenues in which the work described in this thesis can be used for future research.

Coverage The design of a test set which maximizes the coverage obtained when a rule-based system is exercised using that test set is of interest to researchers working in the area of testing rule-based systems. The rule-based systems to be tested may be operating as stand alone systems, embedded in conventional software systems, or operate as a member of a CDPS. We believe that the path model can be applied both to measure the coverage obtained with an existing test set and to aid the designer of a test set in adding new test cases to an existing test set to improve the coverage obtained. Using the path model, one could develop a quantitative measure for coverage, and determine the improvement in coverage obtained on a test set as new test cases are added as well as the path which were not adequately exercised by the test set. New test cases could then be developed by examining the paths that were identified as having not been adequately exercised.

In section 5.2.1 of this thesis, we described three metrics for determining the paths in which all rules have fired, when a rule-based system is exercised on a given test set.

We referred to these metrics as the Conservative, Moderate, and Liberal metrics. In fact, these metrics can be extended to also provide a count of the number of rules in each path that have fired; thus, these metrics could provide a means for measuring the coverage obtained for a given test set, where coverage is evaluated by measuring the degree to which all the paths in a rule base have been tested; that is, we would be able to verify that paths in which all rule have fired did achieve the logical completion specified by the rule base designer. If these metrics are used to measure coverage, they also identify paths in which the number of rules fired is considered to be inadequate.

Each path in which the number of rules fired is considered to be inadequate can be used to aid the designer of the test set in constructing new test cases which increase the coverage of the test set. Additional test cases would be constructed to ensure that the data items required by the rules in paths in which few rules fired will be present in the working memory of the rule-based system. With the addition to the test set of test cases which ensure that the data items required for all the rules to fire in the paths in which only few rules were able to fire in the original test set will be present in the working memory of the system, the coverage of the test set on the rule base is increased.

In section 5.1.1, we explain how to analyze the paths in a rule base to determine the data items required for all the rules in a path to fire. Determining the data items required for all the rules in a path to fire would be an important first step in generating new test cases. However, the entire process of developing a new test case that would adequately exercise the rules in a specific path is still an unsolved problem.

Multiagent Testbed In chapter 3, we indicated that our long term goal was to experiment with a multiagent testbed to determine the effect that an organization chosen for a CDPS has on performance. We believe that based on the Blackbox Expert, a multiagent testbed can be developed that will be instrumental in providing a mechanism for researchers to experimentally validate models they develop for CDPS. We have designed the components of the Blackbox Expert to be reused in the construction of such a multiagent testbed. We believe that such a testbed could be constructed using one of the commercially available blackboard systems, such as

GBB. Then, the CDPS designer would be able to modify different parameters as specified by the organizational model, and measure the computational and functional performance of each agent in the CDPS.

Theoretical Extension The organizational model discussed in this thesis provides a mechanism for describing a CDPS at a high level of abstraction; however, the organizational model is **only** concerned with issues such as the number of agents in the CDPS, the protocols that are used by the agents when planning, and the access privilege of each agent on the blackboard. The path model captures the actions taken by a rule-based system at a low level of abstraction; the path model considers the individual rules that are fired by the rule-based system as it is problem-solving.

In order to provide researchers with a formal method for incorporating the information gained by using the path model to study the intra-agent perspective, we believe that we will require a model, in addition to the organizational model, for the inter-agent perspective. One possibility would be to model an agent as a set of paths, and the computation carried out by an agent in a CDPS could then be modeled by a "sequence of tuples". Each tuple would indicate a path in which all rules have fired and the logical completion achieved by the rules in that path. Let us refer to this sequence as a *line of reasoning*; and let each tuple in a line of reasoning represent a step that is taken by an agent in reaching a final state. Capturing the computation carried out by a rule-based system using a line of reasoning gives us a model that is at a higher level of abstraction than the individual rules considered when constructing paths.

Using a line of reasoning to capture the computation that is to be carried out by each agent in a CDPS, we can determine the data items produced by one agent that are required by another agent, and also *when* the data items are required. We can model each agent in a CDPS using a line of reasoning to indicate the computation that it is performing. We can analyze the paths in these lines of reasoning to determine the data items that are required and data items produced by an agent at each stage of the computation that it would follow when problem-solving. When one agent requires a data item to take the next step in its line of reasoning that has not been produced by any of the previous steps it has taken, and there is another agent in the CDPS

that will produce that particular data item (as indicated by its line of reasoning), then the agent producing that data item should share it with the agent that requires that data item. In addition, we would be able to identify the precise step in which a data item is required by an agent, giving an indication of *when* the data item should be made available.

In the experiment we described in section 5.3.2, we were able to determine paths that were responsible for the occurrence of undesirable events (we identified goals that were achieved more often with a reduced number of data items) as the data items available to a rule-based system were reduced. Once the paths responsible for the occurrence of undesirable events have been detected, the CDPS designer can examine the lines of reasoning that may be attempted by the rule-based systems that include these paths. The CDPS designer can then examine the line of reasoning of each agent in the CDPS and adjust the data distribution, if necessary, to ensure that undesirable events will be avoided.

Bibliography

[1] Leslie Lamport. Time, clocks, and of events in distributed systems. *Communications of the ACM*, 21(7):558–565, 1978.

[2] Alan H. Bond and Les Gasser. An analysis of problems and research in DAI. In *Readings in Distributed Artificial Intelligence*, pages 3–35. Morgan Kaufman Publishers, Inc., San Mateo, California, 1988.

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

[4] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computing Systems*, 3(1):63–75, 1985.

[5] Charles E. McDowell and David P. Helmbold. Debugging concurrent programs. *ACM Computing Surveys*, 21(4):593–621, 1989.

[6] Randall Davis and Reid G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20:63–109, 1983.

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

[8] Sarit Kraus. Agents contracting tasks in non-collaborative environments. In *Proc. Eleventh National Conference on Artificial Intelligence (AAAI 93)*, July 1993.

[9] M. Minsky. *Society of Mind.* Simon and Schuster, New York, 1986.

[10] Keith Decker and Victor Lesser. Quantitative modeling of complex computational task environment. In *Proc. Eleventh National Conference on Artificial Intelligence (AAAI 93)*, July 1993.

[11] J. S. Rosenschien and M. R. Genesereth. Deals among rational agents. In *Proc. Ninth International Conference on Artificial Intelligence (AAAI 85)*, pages 91–99, August 1985.

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

[13] Jeffrey S. Rosenschein. *Rational Interaction Cooperation Among Intelligent Agents.* PhD thesis, Stanford Univ., 1986.

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

[15] Barbara Hayes-Roth. A blackboard architecture for control. *AI Journal*, 26:251–321, 1985.

[16] H. Penny Nii. Blackboard application systems and a knowledge engineering perspective (part 2). *AI Magazine*, August 1986.

[17] Robert Engelmore and Tony Morgan. *Blackboard Systems.* Addison Wesley, Reading, Mass., 1988.

[18] V.R. Lesser and D. Corkill. DVMT: A tool for investigation of distributed problem solving networks. In M.N. Huhns, editor, *Distributed Artificial Intelligence.* Morgan Kaufmann, 1987.

[19] Daniel D. Corkill and Victor R. Lesser. Unifying data-directed and goal-directed control : An example and experiments. In *AAAI*, pages 143–147, 1982.

[20] Stephanie Cammarata, David McArthur, and Randall Steeb. Strategies of cooperation in distributed problem solving. *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 767–770, 1983.

[21] Reid G. Smith and Randall Davis. Frameworks for cooperation in distributed problem solving. *IEEE Transaction of Systems, Man, and Cybernetics*, SMC 11(1):61–70, January 1981.

[22] Gilad Zlotkin and Jeffrey Rosenschein. Negotiation and task sharing among autonomous agents in cooperative domains. In *IJCAI*, pages 912–917, 1989.

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

[24] D. Bobrow. Dimensions of interaction: A shift of perspective in artificial intelligence. *AI Magazine*, 12(3), 1991.

[25] N. Findler and U. Sengupta. An overview of some recent and current research in the AI lab at Arizona State University. *AI Magazine*, 12(3), 1991.

[26] Thomas W. Malone. Modeling coordination in organizations and markets. *Management Science*, 33(10):1317–1332, 1987.

[27] C. Grossner and T. Radhakrishnan. Organizations for cooperating expert systems. In *22nd Southeastern Symposium on System Theory*, March 1990.

[28] R. Clark, C. Grossner, and T. Radhakrishnan. Consensus: A planning protocol for cooperating expert systems. In *11th International Workshop on Distributed Artificial Intelligence, Glen Arbor, Michigan*, February 1992.

[29] R. Clark, C. Grossner, and T. Radhakrishnan. Consensus and Compromise: Planning in cooperating expert systems. *Submitted for review to Int. Journal of Intelligent and Cooperative Information Systems*, 1993.

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

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

[32] J. Galbraith. *Designing Complex Organizations.* Addison Wesley, Reading, Mass., 1973.

[33] C. Grossner, A. Preece, P. Gokulchander, T. Radhakrishnan, and C.Y. Suen. Exploring the structure of rule based systems. In *Proc. Eleventh National Conference on Artificial Intelligence (AAAI 93)*, 1993.

[34] C. Grossner, P. Gokulchander, A. Preece, and T. Radhakrishnan. Revealing the structure of rule-based systems. *Submitted for review to IEEE SMC*, November 1993.

[35] A. Preece, C. Grossner, P. Gokulchander, and T. Radhakrishnan. Structural validation of expert systems: Experience using a formal model. In Jay Liebowitz, editor, *World Congress on Expert Systems.* Macmillan New Media, January 1994. Published on CD-ROM.

[36] C. Grossner, A. Preece, P. Gokulchander, and T. Radhakrishnan. Data sharing among cooperating rule-based systems. In *Submitted for review for the 13th International Workshop on Distributed Artificial Intelligence*, March 1994.

[37] A. Preece, C. Grossner, P. Gokulchander, and T. Radhakrishnan. Structural validation of expert systems: Experience using a formal model. In *Eleventh National Conference on Artificial Intelligence (AAAI 93): Workshop on Validation and Verification of knowledge-Based Systems*, July 1993.

[38] A. Preece, P. Gokulchander, C. Grossner, and T. Radhakrishnan. Modeling rule base structure for expert system quality assurance. In *Thirteenth International Joint Conference on Artificial Intelligence: Workshop on Validation of knowledge-Based Systems*, August 1993.

[39] A. Preece, C. Grossner, P. Gokulchander, and T. Radhakrishnan. Structure-based validation of rule-based systems. *Submitted for review to IEEE Knowledge and Data Engineering*, January 1994.

[40] Paul R. Cohen. A survey of the eighth national conference on artificial intelligence: Pulling together or falling apart? *AI Magazine*, 12(1):16–41, Spring 1991.

[41] P. J. Gymtrasiewicz and E. H. Durfee. Logic of knowledge and belief for recursive modeling: Preliminary report. In *Proceedings of the National Conference on Artificial Intelligence*, pages 628–634, July 1992.

[42] James G. March and Herbert A. Simon. *Organizations.* John Wiley and Sons, New York, 1958.

[43] T. Ishida. The Tower of Babel: Towards organization-centered problem solving. In *Proc. 11th International Workshop on Distributed Artificial Intelligence*, pages 141–154, February 1992.

[44] Moshe Tennenholtz and Yoram Moses. On cooperation in a multi-entity model (preliminary report). In *IJCAI*, pages 918–923, 1989.

[45] B. Chandrasekaran. Generic tasks in knowledge based reasoning: High level building blocks for expert system design. *IEEE Expert*, 1(3):23–30, Fall 1986.

[46] Gregg R. Yost. Acquiring knowedge in SOAR. *IEEE Expert*, 8(3):26–34, June 1993.

[47] Elaine Rich. *Artificial Intelligence.* McGraw Hill, New York, New York, 1983.

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

[49] J.F. Voss and T.A. Post. On the solving of ill-structured problems. In M. Chi R. Glaser and M. Farr, editors, *The Nature of Expertise.* Lawrence Erlbaum Associates, 1988.

[50] A. Newell. Heuristic programming: Ill-structured problems. *Progress in Operations Research*, 3, 1969.

[51] W. Swartout. DARPA workshop on planning. *AI Magazine*, 9(2):101–112, 1989.

[52] Edmund H. Durfee and Victor R. Lesser. Incremental planning to control a blackboard based problem solver. In *AAAI*, pages 58–64, 1986.

[53] Edmund H. Durfee and Victor R. Lesser. Using partial global plans to coordinate distributed problem solvers. In *IJCAI*, pages 875–883, 1987.

[54] Mark Drummond and Ken Currie. Goal ordering in partially ordered plans. In *IJCAI*, pages 960–965, 1989.

[55] D.A. Waterman. *A Guide to Expert Systems.* Addison Wesley, 1986.

[56] Keung-Chi Ng and Bruce Abramson. Uncertainty management in expert systems. *IEEE Expert*, 5(2):29–48, April 1990.

[57] W.A. Woods. What's important about knowledge representation. *IEEE Computer*, 16(10):22–27, October 1983.

[58] S.K. Goyal and W. Worrest. Expert systems in network management and maintenance. In *Proc. ICC*, pages 1225–1229, 1986.

[59] Victor R. Lesser, Daniel D. Corkill, Robert C. Whitehair, and Joseph A. Hernandez. Goal relationships and their use in a blackboard architecture. In V. Jagannathan, Rajendra Dodhiawala, and Lawrence S. Baum, editors, *Blackboard architectures and applications*, volume 3 of *Perspectives in Artificial Intelligence*, chapter 1, pages 9–26. Academic Press, 1989.

[60] D.D. Corkill, K.Q. Gallagher, and K.E. Murray, editors. *GBB: A Generic Blackboard Development System*, Philidelphia, PA., August 1986.

[61] Victor R. Lesser, Daniel D. Corkill, Robert C. Whitehair, and Joseph A. Hernandez. Focus of control through goal relationships. In *IJCAI*, pages 497–503, 1989.

[62] Michael Georgeff. Communication and interaction in multi-agent planning. In *AAAI 83*, pages 125–129, 1983.

[63] Edmund H. Durfee and Victor R. Lesser. Partial global planning : A coordination framework for distributed hypothesis formation. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(5):1167–1183, September 1991.

[64] D. Randall and R.G. Smith. The contract net protocol: High level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, December 1980.

[65] M. Benda, V. Jagannathan, and R. Dodhiawala. On optimal cooperation of knowledge sources - an empirical investigation. Technical report, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, Washington, July 1986.

[66] Norman Abramson. *Information Theory and Coding.* McGraw-Hill, New York, 1963.

[67] R. Clark and C. Grossner. Generation of organizations for cooperating expert systems. DAI Technical Report DAI-0290-0002, Concordia University, Montreal, Quebec, February 1990.

[68] K. Pitula, T. Radhakrishnan, and C. Grossner. Distributed Blackbox: A test bed for distributed problem solving. In *IEEE Int. Phoenix Conference on Computers and Communications*, March 1990.

[69] C. De Koven and T. Radhakrishnan. An experiment in group problem solving. In *Multi-User Interfaces and Applications*, September 1990.

[70] J. Lyons and C. Grossner. A blackbox expert system: User requirements. DAI Technical Report DAI-0190-0001, Concordia University, Montreal, Quebec, January 1990.

[71] D.E. Smith. Controlling backward inference. *Artificial Intelligence*, 39(2):145–208, June 1989.

[72] J. Lyons and C. Grossner. A blackbox expert system: Software requirements specification. DAI Technical Report DAI-0490-0003, Concordia University, Montreal, Quebec, April 1990.

[73] C. Culbert. *Artificial Intelligence Section, Johnson Space Center.* Houston, 1989.

[74] W. Boehm. Software engineering: R and D trends and defense needs. In P. (Ed) Wegener, editor, *Research Directions in Software Technology.* M. I. T. Press, Cambridge, Mass., 1979.

[75] S. Maxwell and H. Delany. *Designing Experiments and Analyzing Data.* Wadsworth Publishing Company, 1990.

[76] T. K. Sellis, N. Rousscpoulos, and R.T. Ng. Efficient compilation of large rule bases using logical access paths. *Information Systems*, 15(1):73–84, 1990.

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

[78] John Rushby and Judith Crow. 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, February 1990. 93 pages.

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

[80] W. B. Gevanter. The nature and evaluation of commercial expert systems building tools. *IEEE Computer*, 20(5), May 1987.

[81] M. A. Shwe, S. W. Tu, and L. M. Fagan. Validating the knowledge base of a therapy planning system. *Methods of Information in Medicine (West Germany)*, 28(1):36–50, January 1989.

Appendix A

Source Tables

Score

Multivariate Tests of Significance

Source of Variance	df	F
Between Group	9	112.8
Within Group	11	

Univariate F-tests for Individual Pairwise Contrasts

	Source of Variance	Sum of Squares	df	Mean Square	F
1 vs 2	Between Group	60528.40	1	60528.40	80.40
	Within Group	14304.60	19	752.87	
2 vs 3	Between Group	11662.23	1	11662.23	22.19
	Within Group	9986.28	19	525.59	
3 vs 4	Between Group	3132.90	1	3132.90	37.55
	Within Group	1585.10	19	83.43	
4 vs 5	Between Group	1863.23	1	1863.23	11.26
	Within Group	3145.28	19	165.54	
5 vs 6	Between Group	1081.60	1	1081.60	10.63
	Within Group	1933.40	19	101.76	
6 vs 7	Between Group	2755.60	1	2755.60	22.24
	Within Group	2354.40	19	123.92	
7 vs 8	Between Group	1716.10	1	1716.10	22.98
	Within Group	1418.90	19	74.68	
8 vs 9	Between Group	3553.23	1	3553.23	41.08
	Within Group	1643.28	19	86.49	
9 vs 10	Between Group	2739.03	1	2739.03	61.63
	Within Group	844.48	19	44.45	

Total Ball Errors

Multivariate Tests of Significance

Source of Variance	df	F
Between Group	9	65.14
Within Group	11	

Univariate F-tests for Individual Pairwise Contrasts

	Source of Variance	Sum of Squares	df	Mean Squares	F
1 vs 2	Between Group	10.00	1	10.00	13.57
	Within Group	14.00	19	0.74	
2 vs 3	Between Group	4.23	1	4.23	12.79
	Within Group	6.28	19	0.33	
3 vs 4	Between Group	2.50	1	2.50	10.56
	Within Group	4.50	19	0.24	
4 vs 5	Between Group	2.50	1	2.50	13.57
	Within Group	3.50	19	0.18	
5 vs 6	Between Group	1.23	1	1.23	7.11
	Within Group	3.28	19	0.17	
6 vs 7	Between Group	3.60	1	3.60	15.55
	Within Group	4.40	19	0.23	
7 vs 8	Between Group	2.03	1	2.03	15.55
	Within Group	2.48	19	0.13	
8 vs 9	Between Group	4.90	1	4.90	13.11
	Within Group	7.10	19	0.37	
9 vs 10	Between Group	5.63	1	5.63	37.17
	Within Group	2.88	19	0.15	

SCORE and Total Ball Errors

Plan Failure Rate					
Multivariate Tests of Significance					
Source of Variance		df			F
Between Group		9			9.41
Within Group		11			
Univariate F-tests for Individual Difference Contrasts					
	Source of Variance	Sum of Squares	df	Mean Squares	F
1 vs 2-10	Between Group	4847.84	1	4847.84	84.72
	Within Group	1087.18	19	57.22	
8 vs 9-10	Between Group	192.53	1	192.53	4.59
	Within Group	797.13	19	41.95	
9 vs 10	Between Group	1254.40	1	1254.40	5.34
	Within Group	4462.60	19	234.87	

Plan Failure Rate

Accesses per Planning Phase					
Multivariate Tests of Significance					
Source of Variance			df		F
Between Group			9		25.10
Within Group			11		
Univariate F-tests for Individual Polynomial Contrasts					
Degree	Source of Variance	Sum of Squares	df	Mean Squares	F
ONE	Between Group	7299173.21	1	7299173.21	296.54
	Within Group	467667.35	19	24614.07	
TWO	Between Group	1527223.11	1	1527223.11	92.92
	Within Group	312294.44	19	16436.55	
THREE	Between Group	1274406.21	1	1274406.21	109.57
	Within Group	220988.04	19	11630.95	
FOUR	Between Group	619474.15	1	619474.15	77.65
	Within Group	151572.72	19	7977.51	
FIVE	Between Group	233600.94	1	233600.94	87.97
	Within Group	50451.07	19	2655.32	
SIX	Between Group	66937.55	1	66937.55	63.08
	Within Group	20163.22	19	1061.22	
SEVEN	Between Group	10123.90	1	10123.90	27.69
	Within Group	6945.79	19	365.57	
EIGHT	Between Group	2448.73	1	2448.73	19.10
	Within Group	2435.69	19	128.19	

Accesses per Execution Phase					
Multivariate Tests of Significance					
Source of Variance			df		F
Between Group			9		55.34
Within Group			11		
Univariate F-tests for Individual Polynomial Contrasts					
Degree	Source of Variance	Sum of Squares	df	Mean Squares	F
ONE	Between Group	33937187.79	1	33937187.79	264.23
	Within Group	2440364.59	19	128440.24	
TWO	Between Group	8954423.30	1	8954423.30	117.92
	Within Group	1442793.28	19	75936.49	
THREE	Between Group	5577427.41	1	5577427.41	85.42
	Within Group	1240524.91	19	65290.78	
FOUR	Between Group	3284869.00	1	3284869.00	74.54
	Within Group	837307.31	19	44068.81	
FIVE	Between Group	1282606.94	1	1282606.94	74.80
	Within Group	325810.47	19	17147.92	
SIX	Between Group	405173.44	1	405173.44	77.28
	Within Group	99620.28	19	5243.17	
SEVEN	Between Group	56963.44	1	56963.44	39.41
	Within Group	27464.79	19	1445.52	

Blackboard Accesses: Planning and Execution

Rules Fired per Planning Phas					
Multivariate Tests of Significance					
Source of Variance			df		F
Between Group			9		64.562
Within Group			11		
Univariate F-tests for Individual Polynomial Contrasts					
Degree	Source of Varience	Sum of Squares	df	Mean Square	F
ONE	Between Group	53289.78	1	53289.78	264.80
	Within Group	3823.67	19	201.25	
THREE	Between Group	3846.61	1	3846.61	65.47
	Within Group	1116.39	19	58.76	
FIVE	Between Group	570.02	1	570.02	22.79
	Within Group	475.20	19	25.01	

Rules per Execution Phase					
Multivariate Tests of Significance					
Source of Variance			df		F
Between Group			9		37.57
Within Group			11		
Univariate F-tests for Individual Polynomial Contrasts					
Degree	Source of Variance	Sum of Squares	df	Mean Squares	F
ONE	Between Group	18584.19	1	18584.19	333.90
	Within Group	1057.50	19	55.66	
TWO	Between Group	3727.56	1	727.56	183.54
	Within Group	385.88	19	20.31	
THREE	Between Group	1400.96	1	1400.96	53.86
	Within Group	494.19	19	26.01	
FOUR	Between Group	1219.22	1	1219.22	78.39
	Within Group	295.52	19	15.55	
FIVE	Between Group	147.91	1	147.91	26.55
	Within Group	105.86	19	5.57	
SIX	Between Group	126.26	1	126.26	16.15
	Within Group	148.59	19	7.82	

Rules Fired: Planning and Execution

Appendix B

Glossary of Symbols

A: Attribute vector describing current state of a problem.

$\hat{A}$: An estimate of A.

A'_t: Current state of SP_t: $A'_t \subseteq \hat{A}$.

A''_j: Set of factors whose instances are visible in a window when they are stored in region bbr_j.

$AREA(x)$: Function giving area of region x.

a_i: A factor describing a characteristic of a problem state.

$\alpha(t_i, s_j)$: Mapping on S: $\alpha : T \times S \rightarrow S$ where $\alpha(t_i, s_j) = s_k$.

$\hat{\alpha}(t_i, s_j)$: An estimate of $\alpha(t_i, s_j)$: a mapping on S; $\alpha : T \times S \rightarrow S$ where $\alpha(t_i, s_j) = s_k$.

$\mathcal{A}^{r_i}$: RHS of a rule, set of facts.

$\mathcal{B}$: Set of Predicates.

B^C: The number of correctly located mappable balls.

B_E: The number of ball errors.

B^M: The total number of mappable balls.

b^T: The total number of entry/exit positions of the Blackbox.

b^V: The total value of the beams fired to solve the puzzle.

B^W: The number of incorrectly positioned balls.

BB: Model of a blackboard.

bbr: Begin on a blackboard.

$\beta \Rightarrow^+ g_k$: Set of subproblems β that need to be solved to achieve g_k.

$C\{rbs_1, rbs_2, \ldots, rbs_n\}$: A set of rule-based systems that plan using Consensus.

CP: A capability matrix.

CP_k^t: Completion predicates for a path.

CS: Coordination structure of an organization.

CT_k^t: Completion templates for a path.

$CT_{i,k}^t$: Set of templates on the LHS of r_i that are not satisfied by the rules in path k of task t.

CW: Combined window of a planning group.

$DE(rbs_i/SP_t)$: Information deficit of a rule-based system rbs_i in an organization when it solves SP_t.

$DP(rbs_i/PG_k)$: Information deficit of a rule-based system rbs_i in an organization when planning as a member of planning group PG_k.

$\mathcal{D}\{rbs_i : PG - rbs_i\}$: A set of rule-based systems that plan using Decree.

DM: The dimension of one level of a blackboard.

δ: Most general unifier, $\langle \Lambda_i, L_i \rangle \cdot \delta = \langle \lambda_i, l_i \rangle$.

E_A: The average number of accesses to the blackboard during an execution phase.

E_i: Set of subproblems executed by the problem solver to achieve a plan.

E_R: The average number of rules fired during an execution phase.

$\mathcal{E} = \langle E, RB, WM \rangle$: Rule-based system: E is an inference engine, RB is a rule base, and WM is working memory.

ES: Set of rule-based systems in an Organization.

F_k^t: A fragment k in task t.

FC: Set of all factors visible in a window.

FD: Set of factors that may be stored on a level of a Blackboard.

FX_i: Fixation of factor a_i.

FPC_t: Decision-factor set for SP_t.

$f_i = \langle \lambda_i, l_i \rangle$: A fact used by a rule-based system: $\lambda_i \in R$.

G: The set of goals used by the problem solver.

G': The set of goals from which the rule-based systems in an organization can select goals while planning: $G' \subseteq G$.

g_i: A goal.

GPS_t: A group planning set.

γ: Set of goals chosen by the problem solver when planning: $\gamma \subseteq G$.

$H(a_i)$: Entropy.

$\mathcal{I}^{r_i}$: LHS of a rule, set of templates.

$\mathcal{IP}(a_i/w_j)$: Information potential of factor $a_i$in window w_j.

Λ_i: Specification for a predicate.

LV: Set of levels on the blackboard.

L_i: List of variables.

L^C: The number of grid squares which do not contain a mappable ball that are correctly identified.

L^T: The total number of grid squares which do not contain mappable balls.

lv_i: A level on a blackboard.

L^W: The number of grid squares which do not contain a mappable ball that are incorrectly identified.

l_i: Data elements.

LB: The set of lower bounds for a level of a blackboard.

lb_j: A lower bound for the range of a level of a blackboard.

LBR: The set of lower bounds for a region.

lbr_j: A lower bound for the range of a region.

$\mho$: A mapping from a set Templates, or a set of facts, to a set of predicates.

P_A: The average number of accesses to the blackboard during a planning phase.

P_F: the plan failure rate.

P_R: The average number of rules fired during a planning phase.

P_k^t: A path k in task T_t.

$\hat{P}$: A problem where $\hat{A}$, $\hat{\alpha}(t_i, s_j)$, $\hat{T}$, and $\hat{S}_F$ have been estimated by the problem solver.

P^I: Problem solver's view of an ill structured problem: $P^I = (\hat{P}, P^S, G)$.

P^L: A plan.

P^S: Set of subproblems for $\hat{P}$.

PC_t: Constraint set for SP_t.

PG_k: A planning group.

PP_i: A partial plan.

PS^I: The problem-solving process for an ill-structured problem.

$PT^t_{i,k}$: Set of templates on the LHS of r_i that are satisfied by the rules in path k.

Φ: The set of rules in a path.

Ψ: An organization of rule-based systems.

R: The set of all predicates used by a rule-based system.

RG: Set of regions in a window.

r_i: A rule.

r^f_i: A concrete rule firing:

$r_i \prec r_j$: r_j depends upon r_i.

$r^f_i \gg r^f_j$: r^f_i causes r^f_j to fire.

$r^f_i \mapsto r_j$: An unambiguous (unequivocal) mapping from the concrete rule firing r^f_i to the abstract rule r_j.

$r^f_i \overset{e}{\mapsto} r_j$: An equivocal mapping from r^f_i to r_j.

rbs_i: A rule-based system.

S: Set of states in the state space of a problem.

S'_t: State space of SP_t: $S'_t \subseteq S$.

s_i: A state of a problem.

S^F: Set of states which are acceptable solutions to a problem.

$\hat{S}^F$: An estimate of S^F: the set of states which are acceptable solutions to a problem.

S^I: Set of initial states for a problem.

SC_i: A zero memory information source.

SP_t: A subproblem of P^I.

$SP_t \rightsquigarrow U$: set of rules U that assert facts using all the predicates of a logical completion for SP_t.

SP^t_k: Start predicates of a path.

SR^t_k: Start rules in a path.

ST^t_k: Start templates of a path.

$PT^t_{i,k}$: Set of templates on the LHS of start rule r_i that are satisfied by the rules in path k.

T: Set of transformations used to traverse the state space of a problem.

$\mathcal{T}$: A thread.

T'_t: Set of transitions for SP_t: $T'_t \subseteq \hat{T}$.

T_t: Rules used to solve SP_t.

T^L: Set of all transformations on A that are legal.

T^P: Set of all transformations on A that are possible, but not legal.

$\hat{T}$: An estimate of T.

$\langle \Lambda_i, L_i \rangle$: A template.

UB: The set of upper bounds for a level of a blackboard.

ub_j: An upper bound for a level of a blackboard.

UBR: The set of upper bounds for a region.

ubr_j: An upper bound for a region.

$V \rightarrow r_j$: A rule r_j is reachable from a set of rules V.

v_j: An instance of a factor a_i.

w_i: Window onto a blackboard.

$W \hookrightarrow r_j$: W is an enabling-set for r_j.

WS: The set of windows in an organization.

Z_t: End predicates for Task t.